ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO
UNIVERSIDAD

DECANATURA DE INGENIERÍA INDUSTRIAL
MAESTRÍA EN INFORMÁTICA
FORMATO DE ENTREGA TRABAJO DE GRADO

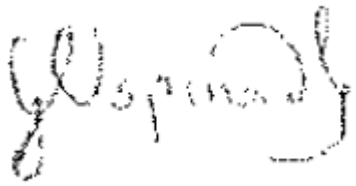
Fecha de entrega:10 de abril del 2023

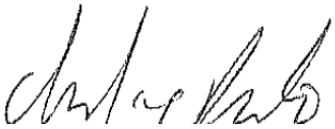
Estudiante: Charlie Alberto Angulo

Director: Gerardo Ospina Hernández

El presente documento avala la entrega del trabajo de grado por parte del director y codirector.

Documentos anexos: copia digital del Trabajo de Grado (1).





Firma Director **Firma Estudiante**

ENTORNO EXPERIMENTAL DE PROCESAMIENTO DE DATOS DISTRIBUIDOS INTEGRANDO DEVOPS, BAJO UN DESPLIEGUE DE INFRAESTRUCTURA COMO CÓDIGO EN LA NUBE.

Trabajo de grado para optar al título de
Magister en Informática.

CHARLIE ALBERTO ANGULO ANGULO

ESCUELA COLOMBIANA DE INGENIERÍA
JULIO GARAVITO
MAESTRIA EN INFORMÁTICA
FACULTAD DE INGENIERÍA
BOGOTA D.C
2022

Dedico este trabajo principalmente a Dios, quien siempre ha estado presente en momento de mi vida, bendiciéndome y guiándome en el camino de alcanzar mis sueños. A mi madre por su amor, y esfuerzo por sacarme adelante, por qué gracias a ella estudie esta carrera y me demuestra cada día lo orgullosa que esta ella de mí. A mi esposa que siempre estuvo ahí apoyándome y aconsejándome para lograr este sueño.

AGRADECIMIENTOS

En primer lugar, a nuestro padre celestial, que me ha acompañado en cada momento e instante de mi vida, guiándome motivándome y dándome fuerza para superarme y lograr todo lo he obtenido hasta este momento de mi vida independientemente las situaciones vividas.

RESUMEN

En esta investigación se muestra la implementación y puesta en marcha de una arquitectura orientada a la experimentación y práctica en procesamiento de datos distribuido con la opción de integrar un flujo de entrega continua de software con *devops*; mediante un despliegue automatizado de infraestructura como código en la nube. Permitiendo obtener un entorno experimental para estudiantes interesados en asumir cargos laborales *Cloud Engineering* y *Data Engineering*, logrando minimizar el tiempo y el esfuerzo que se consume en la configuración del clúster, promoviendo el aprendizaje práctico y experimental de tecnologías demandadas en el mercado.

Palabras Clave: Data processing, big data, devOps, hadoop, spark, terraform, data, cloud. aws, clúster.

ABSTRACT

This implementation and commissioning of an architecture oriented to experimentation and practice in distributed data processing with the option of integrating a continuous software delivery flow with devops; through an automated deployment of infrastructure as code in the cloud. Allowing to obtain an experimental environment for students interested in assuming Cloud Engineering and Data Engineering job positions, minimizing the time and effort consumed in the configuration of the cluster, promoting practical and experimental learning of technologies demanded in the market.

Keywords: Data processing, big data**,** *devOps*, hadoop, spark, data, *cloud*, aws, cluster, infrastructure as code.

TABLA DE CONTENIDO

LISTA DE FIGURAS

ACRONIMOS

PaaS: Plataforma como Servicio

IaaS: Infraestructura como Servicio

SaaS: Software como servicio

IasC: Infraestructura como código

DaaS: Datos como servicio

CI: Integración continua

CD: Despliegue continuo

CT: Prueba continua

CM: Monitoreo continuo

AWS: Amazon Web Services

GCP: Google Cloud Platform

Vpc: Red virtual privada en la nube.

(AM): Administrador de aplicaciones de yarn.

(SCP): Herramienta de Linux que permite el copiado de archivos cifrados.

GLOSARIO

1.1 Cloud Engineer:

Un ingeniero de la nube es un profesional de TI responsable de cualquier tarea tecnológica asociada a la computación en nube, incluidos el diseño, la planificación, la gestión, el mantenimiento y el soporte.(*What Is a Cloud Engineer and How Do You Become One?*, n.d.)

1.2 Data Engineer:

Los ingenieros de datos son una parte fundamental en cualquier proceso de ciencia de datos. Son perfiles muy demandados en cualquier entorno donde se manejen datos. Una data *engineer* es aquel profesional enfocado en el diseño, desarrollo y mantenimiento de los sistemas de procesamiento de datos dentro de un proyecto de Big data.(Bello, 2022)

1.3 Big data:

Es la agrupación de múltiples tendencias tecnológicas, el termino describe grandes volúmenes de datos generados de diferentes fuentes de información o un grupo de registros de datos muy grandes y complejos que resultan difíciles de procesar utilizando las aplicaciones tradicionales de procesamiento de datos o las herramientas de los sistemas de bases de datos disponibles. (Rida 2020)

1.4 Teorema CAP:

El teorema CAP o teorema *Brewer*, dice que en sistemas distribuidos es imposible garantizar a la vez: (consistencia): al realizar una consulta o inserción siempre se tiene que recibir la misma información, (disponibilidad) todos los clientes puedan leer y escribir, aunque se haya caído uno de los nodos y (tolerancia a particiones) el sistema tiene que seguir funcionando, aunque existan fallos (ConsistencyAvailability-Partition Tolerance).(Ali et al., 2015)

1.5 Visualización:

Es la forma de como mostrar y comunicar los datos de forma eficaz en diferentes entornos se ha convertido cada vez más en un importante contenido de investigación el uso de métodos de visualización como gráficos, imágenes y animaciones. (Cao, Lin, and Ma 2020)

1.6 Arquitectura de software:

La arquitectura de las aplicaciones de *software* es el proceso de definición de una solución estructurada que satisface todos los requisitos técnicos y operativos, al tiempo que optimiza atributos de calidad comunes como el rendimiento, la seguridad y la capacidad de gestión.(Zhelev & Rozeva, 2017)

1.7 Big Data en On-Premise:

En este caso hablamos de fijar el *Big Data* en las instalaciones propias, en el hardware propiedad de la empresa. Será un hardware dedicado en exclusiva al procesamiento de estos datos y que debe estar preparado para dar respuesta a todos los procesos en el tiempo establecido.

1.8 Big Data en la Nube/Cloud:

Esta opción cuenta con una infinidad de funcionalidades y grandes ventajas añadidas. Uno de los puntos fuertes es la inexistencia prácticamente de una inversión en hardware. Además, está altamente automatizado y personalizado y con poco presupuesto se puede iniciar fácilmente.

1.9 Big Data Híbrido:

Une los mejor de *big data on-premise y cloud*, ya que es un sistema flexible, no pierdes el control de los datos, pero no necesitas una grandísima infraestructura como en el *On-Premise*.

1.10 Computación en la nube

Es un término usado para referirse a un modelo de infraestructura en el cual un usuario puede acceder a un servicio o aplicación bajo demanda y de forma remota.(Osorio et al., 2006)

1.11 Nube Pública:

La infraestructura de la nube se pone a disposición del público en general o de un gran grupo industrial y es propiedad de una organización que vende servicios en la nube.

1.12 Nube Privada:

La infraestructura de la nube se opera únicamente para una organización. Puede ser administrado por la organización o un tercero y puede existir dentro o fuera de las instalaciones. (Osorio et al., 2006)

1.13 Nube Híbrida:

Es la combinación de uno o más entornos de nube pública y privada, de tal forma que las organizaciones empresariales se benefician de las ventajas que proporcionan los dos tipos de infraestructura *cloud*. Así, se dispone de un conjunto de recursos virtuales gestionados por software de administración y automatización que permite a los usuarios acceder a lo que necesiten. (Osorio et al., 2006)

1.14 Plataforma Como Servicio (PaaS):

Ofrece una solución completa para la construcción y puesta en marcha de aplicaciones y servicios Web que estarán completamente disponibles a través de Internet. Algunos ejemplos: *Google App Engine*. (Osorio et al., 2006)(Ashraf, 2014)

1.15 Software Como Servicio (SaaS):

Consiste en la distribución de software donde una empresa proporciona el mantenimiento, soporte y operación que usará el cliente durante el tiempo que haya contratado el servicio. Ejemplos: Gmail. (Osorio et al., 2006)(Ashraf, 2014)

1.16 Infraestructura Como Servicio (IaaS):

Proporciona al cliente una infraestructura de computación como un servicio, usando principalmente la virtualización. El cliente compra recursos a un proveedor externo, para hosting, capacidad de cómputo, mantenimiento y gestión de redes, etc. Ejemplos: Amazon EC2.

1.17 Infraestructura Como Código (IaC):

La infraestructura como código (IaC del inglés Infrastructure as Code) permite gestionar y preparar la infraestructura a través del código, en lugar de hacerlo mediante procesos manuales.

Con este tipo de infraestructura, se crean archivos de configuración que contienen las especificaciones que esta necesita, lo cual facilita la edición y la distribución de las configuraciones. Asimismo, garantiza que usted siempre prepare el mismo entorno. La infraestructura como código codifica y documenta sus especificaciones para facilitar la gestión de la configuración, y le ayuda a evitar los cambios ad hoc y no documentados.(*¿Qué Es La Infraestructura Como Código - Infrastructure as Code?*, n.d.) (Artac et al. 2017)

1.18 Datos como servicio (DaaS):

Esto permite acceder a datos en varios formatos y de múltiples fuentes a través de servicios de los usuarios de la red. Los usuarios podrían, por ejemplo, manipular los datos remotos como operar en un disco local o acceder a los datos de forma semántica en Internet. (Rajesh et al., 2012)

1.19 DevOps:

Conjunto de prácticas que trata de salvar la brecha entre desarrolladores y operaciones cubriendo los aspectos que limitan la entrega de software rápida, optimizada y de calidad, utilizando una serie de principios facilitando la entrega continua de software, pruebas unitarias, retroalimentación continua y reaccionar al cambio eficientemente.(Technology, 2015)

1.20 Entrega continua (Continuous Delivery):

Trata de optimizar la gestión de la infraestructura y la necesidad crítica de equilibrar el tiempo y los recursos. (Technology, 2015)Permitiendo en los equipos de desarrollo realizar entregas de

software en ciclos de vida cortos, para así generar valor en entregas de mínimo productos viables (MVP), liberando software en cualquier momento de forma confiable.

1. INTRODUCCIÓN

1.1 Contexto:

En la actualidad se generan miles de millones de datos, *big data* es el termino describe en la generación grandes volúmenes de datos permitiendo una gran cantidad, variedad, a una gran velocidad, en donde se logra identificar la veracidad y aportar valor a través de los mismos, por esta razón requieren nuevas formas de procesamiento para permitir una mejor toma de decisiones a las organizaciones, facilita el descubrimiento de información valiosa, logrando incluso a optimizar muchos procesos.

Gracias a la analítica de datos altamente avanzada, a menudo impulsada por técnicas de IA, podemos dar sentido y trabajar con flujos de datos enormemente complejos y variados.(Howard, 2020) A medida que los datos se trasladan al ecosistema de la nube, los datos y la analítica componibles son cada vez más relevantes y el enfoque más preferido a la hora de crear aplicaciones analíticas. Los datos y la analítica componibles se refieren al uso de múltiples soluciones de datos, analítica e inteligencia artificial para aumentar la conectividad entre los datos y las acciones empresariales y construir experiencias más flexibles con mayor facilidad de uso. Según *Gartner*, los datos y la analítica componibles promueven la productividad, la velocidad, la colaboración y las capacidades analíticas de las empresas.(*Gartner*, 2021)

En el año (2022), *Gartner* destaca la capacidad de las herramientas de análisis e inteligencia empresarial para ofrecer información automatizada, de modo que el consumidor ideal de las plataformas es el responsable de la toma de decisiones basada en datos, más que el analista o el científico de datos: "Muchas plataformas están añadiendo capacidades para que los usuarios puedan componer fácilmente flujos de trabajo y aplicaciones de automatización de bajo código o sin código.(*Gartner*, 2022)

A medida que las empresas son cada vez más conscientes del valor de las decisiones basadas en datos y del poder de los mismos, crece la relevancia de la visualización de datos y los criterios de presentación de la información a través de cuadros de mando donde *Microsoft Power BI* es el líder del Cuadrante Mágico 2022 de *Garnert*. (Gartner, 2022)

Cada año seguimos viendo cómo la industria evoluciona y acelera la capacidad de ofrecer software con más velocidad y mejor estabilidad. Las organizaciones que se someten a una

transformación *devops* mediante la adopción de la entrega continua son más propensas a tener procesos de alta calidad, bajo riesgo y rentables.(Smith et al., 2021)

Ahora bien, la introducción de la computación en la nube trajo consigo una serie de nuevos retos de ingeniería. Los desarrolladores de software que se enfrentan a con estos retos tienden a converger en soluciones similares basadas en buenas prácticas de diseño. Uno de los patrones y prácticas más se utilizada en la nube actualmente es la infraestructura como código. (Sousa et al., 2021)

Por ello, las empresas necesitan especialistas que sean capaces de digerir esos datos, convertirlos en información y seleccionar cuál es útil y cuál no, para la posterior toma de decisiones utilizando herramientas para la manipulación de datos, en entornos emergentes como la nube y en tecnologías disruptivas como lo son infraestructura como código y *devops*. Un tema importante es el cómo los estudiantes que quieren enfocar su carrera al desarrollo en cargos como *Cloud engineer* (Ingenieros en la nube), *Data Engineering*(Ingenieros de datos) puedan adquirir estas habilidades de manera práctica y aplicando sus conocimientos teóricos en un entorno experimental en la nube, evitando realizar configuraciones que en la mayoría de veces terminan demandando tiempos en exceso y no se logra centrar en el objetivo principal de interactuar, desarrollar y experimentar con estas herramientas.

En esta tesis se propone un ambiente que permita evitar las configuraciones extensas, sacando provecho e integrando los mejores beneficios que ofrece en una plataforma de procesamiento de datos distribuidos en un entorno *cloud* mediante un despliegue de infraestructura como código, manipulando herramientas dominantes en el mercado y utilizadas en los ambientes laborales. En esta misma línea este entorno permitirá la opción de aplicar conceptos o términos que en los últimos años han sonado mucho y están adoptando las organizaciones como lo es *devops*, permitiendo la entrega continua en el ciclo de entrega de software. De esta forma los desarrolladores *cloud* tendrán una experiencia práctica a la hora de interactuar en el mundo de los datos, la nube, infraestructura como código y algunas herramientas utilizadas en *devops*, con un conocimiento mucho más sólido, enfocados en desarrollo y no en las configuraciones complejas, obteniendo así habilidades demandados laboralmente en el mercado y viviendo un símil de lo se encontrarán en un entorno laboral.

1.2 Planteamiento del problema

El descubrimiento de conocimiento y la toma de decisiones a partir de datos es un reto para las empresas en términos de almacenamiento, administración y procesamiento. (Bibri and Krogstie 2019) De acuerdo con *Gartner* Inc. Por lo que los datos resultan ser una herramienta de competitividad que permite a las organizaciones descubrir conocimiento empresarial con el objetivo de incrementar el rendimiento del negocio, y obtener así una ventaja competitiva con respecto a sus competidores. Las tecnologías más innovadoras saldrán reforzadas de la crisis de COVID-19 y entre esas tecnologías se encuentra en primer lugar la Analítica de datos. (Howard, 2020)

La nube sin duda alguna ha sido un aliado potencial para que toda es manipulación, almacenamiento y procesamiento de los datos se dé y continúe creciendo, por lo que los proveedores en la nube juegan un papel clave para las empresas con todo su portafolio de servicios en el mundo de *los datos.*

Por otro lado, la forma en que las empresas entregan el software está cambiando constantemente a medida que el entorno varia, las necesidades del mercado y la tecnología cambian continua y rápidamente, hay más presión para adaptarse a las necesidades del mercado y entregar rápidamente.(*Technology*, 2015) Con el creciente desarrollo de *software*, se está teniendo mucho más en cuenta del desarrollo seguro de código, la infraestructura, el cómo desplegar y realizar el escalamiento de aplicaciones. Por lo ha abierto la puerta a utilizar *devops* en las organizaciones, en la búsqueda de unir el desarrollo con las operaciones para la gestión, despliegue y soporte de software. Desde la necesidad de negocio, las formas de pasar de la integración continua a la entrega continua y sus beneficios.

Hablando de procesamiento de datos distribuidos, la nube y *devops*, existe en el mercado una gran necesidad de talento con habilidades en estos temas pos pandemia que tienen las organizaciones, cargos de desarrollo de software, *Cloud Engeniering* y *Data Engenieering* son los más demandados e inclusive los mejores pagos, por ello las empresas necesitan especialistas que sean capaces de digerir esos datos, limpiarlos, convertirlos en información seleccionada para posteriormente tomar decisiones utilizando herramientas para la manipulación de datos, como lo son *Hadoop*, *Spark, Python, Scala, Sql*.(宗成庆, 2021) Con

conocimiento práctico y experimental en plataformas en la nube, sistemas operativo *Linux*, gestión de redes *cloud*, programación, gestión de bases de datos. entre otras.

El problema es que se carece de un entorno experimental de procesamiento de datos distribuidos basado en un despliegue ágil, que permita evitar al máximo las configuraciones, de alguna manera minimizando gastos utilizando herramientas open *Source* y *Cloud*, permitiendo enfocar el conocimiento experimental y práctico, con la opción de aplicar conceptos o términos utilizados en las organizaciones como lo es *devops* en el de entrega de software. De esta forma los estudiantes que quieran enfocar su carrera al mundo de desarrollo en cargos como mencionados anteriormente tendrán una experiencia práctica a la hora de interactuar en el mundo de los datos y la nube, con un conocimiento mucho más sólidos, enfocados al desarrollo y no en las configuraciones complejas; obteniendo así *habilidades* demandados laboralmente en el mercado, siendo laboralmente competitivos y finalmente viviendo un símil de lo se encontrarán en un entorno laboral.

1.3 Formulación del problema

¿Cómo integrar en un entorno experimental de procesamiento de datos distribuidos, con la opción de almacenamiento, análisis y visualización de datos, adoptando herramientas *devops* en el ciclo de entrega continua de software, basado en una arquitectura de despliegue como código en la nube, para que los estudiantes puedan desarrollar habilidades técnicas en estas tecnologías?

2. OBJETIVOS

2.1 General

Implementar un entorno para la práctica y experimentación de procesamiento de datos distribuidos, bajo una arquitectura de infraestructura como código en la nube, permitiendo realizar a los estudiantes con enfoque en Cloud y Data Engineering; el almacenamiento, procesamiento, análisis y visualización de datos distribuidos, adoptando el ciclo de entrega continua de *software* con herramientas *devops*.

2.2 Objetivos específicos

- Estudiar los entornos y las arquitecturas de procesamientos de datos en la nube.
- Diseñar una arquitectura de procesamiento de datos distribuidos integrado una capa de visualización y herramientas *devops,* bajo un modelo de despliegue como código.
- Implementar la arquitectura diseñada.

  Probar el entorno realizando un procesamiento de datos de un caso de estudio con estudiantes.

3. ALCANCE DEL PROYECTO

La solución planteada está enfocada a los estudiantes interesados en asumir cargos laborales *Cloud Engineering* y *Data Engineering*, minimizando el tiempo y el esfuerzo que se consume en la configuración del clúster para el almacenamiento y procesamiento de datos distribuidos. Promoviendo el aprendizaje práctico y experimental de tecnologías demandadas en el mercado. Se propone realizar la configuración e implementación y el aprovisionamiento de un clúster de procesamiento, almacenamiento, análisis, y visualización de información distribuida mediante un despliegue de infraestructura como código en la nube (IaC), integrando herramientas líderes en el mercado para el procesamiento de datos y la opción de incluir herramientas *devops,* para que los estudiantes interactúen con estas herramientas en un entorno que desencadene un conjunto de acciones de validaciones de código, cobertura, luego de que los estudiantes elijan la opción de versionar el código, en la búsqueda de ofrecerle experimentar un símil de lo que se encontraran, cuando realicen desarrollos en espacios colaborativos laboralmente.

4. PROBLEMA Y MOTIVACIÓN

Algunos problemas a los que se enfrenta un estudiante que requiere configurar un entorno de procesamiento de datos distribuidos localmente.

Problemas		Solución
Especificaciones del Pc	La configuración de un entorno de procesamiento de datos distribuidos requiere, una alta demanda de memoria *ram*, procesadores, almacenamientos, por estos deberán tener buenas características. Adicional una versión de un sistema operativo especifico y habilitar el *Hiper-v* si se va a virtualizar.	Usualmente se resigna a la compra de un pc, memoria *ram* u cambio de procesador o en su defecto a cambio de *board* para habilitar el *hyper-v*. Ahora el estudiante que no está muy interesado con el tema desiste de la idea de aprender.
Configuración de varias herramientas y el *clúster*	Esta es la tarea más compleja porque hay que descargar y configurar *Hadoop*, *Python*, *scala*, *spark*, *jdk* de *java*, *Ide* de desarrollo. En cuanto a el *clúster* parametrizar la especificar la comunicación los nodos, y en la mayoría de las ocasiones se tienen inconvenientes como: No levantan los mismo. No funciona el formateo del *fileSystem* de *hadoop* u spark, No se	Se suele llevar varios días de acuerdo a la disponibilidad para configurarlas, y tratando de ir resolviendo los errores que vallan apareciendo en el camino.

	despliega el *resource* manager u *Yarn* en *hadoop*, no responde el clúster por las especificaciones del *pc*, no inicia el *spark sesión*.	
Parametrización de variables	Desconocimiento en configuración de variables de entornos requeridas.	Investigar, u preguntar en comunidades de desarrollo.
Velocidad de procesamiento	Dado a que es una sola máquina y así se virtualice bien sea con *docker* u otras herramientas, tenemos recursos limitados y la velocidad va a variar de acuerdo a las especificaciones del pc y la cantidad de datos. En muchas ocasiones secuestra el pc y no se puede realizar ninguna otra acción en el mismo debido a que consume la mayor parte de memoria y procesador, incluso generando el famoso error "No responde"	En la mayoría de los casos se deja ejecutando en un lapso de tiempo en el que no se valla a utilizar el computador, esperando a que no aparezca errores con las tareas, en el caso de que aparezcan relanzar el proceso nuevamente.
Visualización interactiva	El poder graficar en el proceso de análisis no es tan fácil con los IDE de desarrollo con los cuales se opta la mayoría de las veces, como interfaz.	Usualmente descargan *plugins* o se migra la información a otra herramienta para lograr graficarla.

Si hablamos de un entorno de procesamiento de datos en la nube. Se tendría que capacitar y profundizar en los servicios que ofrecen los proveedores en la nube, realizando un análisis de costo beneficio, como es su implementación, como funciona, que arquitectura utiliza, y otros temas que consumen tiempo y hace que en la mayoría de los casos desistamos del interés que teníamos de aprender de estas nuevas tecnologías, por lo tedioso que se vuelve configurar el entorno. Si analizamos las soluciones locales como en la nube, le demandan al usuario demasiado tiempo, el usuario que en este caso es el estudiante simplemente quiere afianzar su conocimiento teórico con el práctico y experimental, en un entorno que le permita, recibir datos, limpiarlos, analizarlos, representarlos.

Teniendo en cuenta lo expuesto anteriormente y como mencionábamos en el planteamiento del problema, es necesario implementar un ambiente experimental de procesamiento de datos distribuidos basado en un despliegue ágil, enfocado al desarrollo y no en las configuraciones complejas; obteniendo así habilidades demandados laboralmente en el mercado, siendo laboralmente competitivos y finalmente viviendo un símil de lo se encontrarán en un entorno laboral.

5. ESTRUCTURA DEL PROYECTO

Para el diseño e implementación de este entorno, se definieron las siguientes fases, las mismas en las que está estructurado el documento.

5.1 Fase 1: Análisis

En la fase de análisis el objetivo es revisar la literatura mediante una búsqueda sistémica.

5.2 Fase 2: Planeación y Diseño

En esta fase se realizará el diseño de la arquitectura de referencia para el procesamiento de datos distribuidos integrada con herramientas *DevOps* mediante *IaC*.

5.3 Fase 3: Configuración

En la fase de ejecución se implementará la arquitectura propuesta.

5.4 Fase 4: Ejecución y Diagnóstico

Se colocará a prueba el despliegue de la infraestructura en la nube con todas las configuraciones de las herramientas a utilizar.

5.5 Fase 5: Pruebas y Análisis de resultados

Se probará el entorno completo implementando un caso de uso, en búsqueda de probar el ambiente, recolectando un set de datos, con las métricas del resultado de las ejecuciones.

5.6 Fase 5: Cierre

Conclusiones, trabajos futuros y entregables.

6. ESTADO DEL ARTE

En temas de entornos de procesamiento de datos distribuidos para experimentación, existen algunos ambientes con implementaciones de arquitecturas lideradas por big data similares pero no a plenitud con la que queremos proponer, hay muchos acercamientos con respecto a componentes que se quieren trabajar en este trabajo, por ejemplo con spark tenemos un esquema de utilización de recursos adaptable para cargas de trabajo de Big Data en entornos de contenedores aprovechando la elasticidad vertical de *Docker*, realizando experimentos con apache *spark* y *hadoop* en la nube.(Choi, Cho, and Kim 2021), llevando a cabo la implementación de un *clúster* Spark en máquinas virtuales como en *docker* con un único nodo, cada máquina virtual tiene su propio sistema operativo, así como su propio espacio de trabajo de procesamiento de datos *Spark*

para gestionar los archivos de ejecución. Pero está enfocada a investigar la diferencia de rendimiento de ambas implementaciones.

Con respecto a un ambiente experimental encontramos una plataforma big data, basada en contenedores *docker*, utilizando herramientas como *redis* y *nginx*. (Ruijun 2020) Propone establecer una arquitectura avanzada en la nube, que puede construirse utilizando servidores recién adquiridos o servidores propios de la universidad. La plataforma gestiona el clúster de servidores y divide los recursos mediante tecnología de virtualización, carga las imágenes experimentales de *big data* en la máquina virtual dividida y sirve como entorno del sistema experimental para los estudiantes.

Encontramos también una plataforma de aplicación de ciencia de datos y análisis de *big data* basada en la arquitectura de micro servicios para la educación en el campo de la investigación no profesional, utilizando *Docker*, *JupyterHub*, *spark*, *hdfs*, y *spring boot* para la parte *web*. (Miao et al. 2020) Ayuda a compartir datos, potencia de cálculo y recursos de infraestructura. Además, permite a los usuarios utilizar un entorno más amigable para registrar y compartir el proceso experimental y el modelo visualizado. Utiliza *JupyterHub* y el modelado visualizado como sus dos aplicaciones principales. Los componentes del entorno de infraestructura de la plataforma son el componente de *big data*, la plataforma utiliza micro-servicios como dependencia técnica; en particular, utiliza *Docker*. El sistema de archivos distribuidos del entorno utiliza *HDFS* (*Hadoop Distributed File System*. La arquitectura de computación paralela utiliza *Apache Spark*, se utiliza *Jupyter Notebook* como entorno de ingeniería de código del usuario. Y *JupyterHub* gestiona un entorno de cuaderno multi usuario de forma unificada. Esta solución se acerca a lo que se quiere lograr, pero utilizaremos la construcción y el despliegue mediante infraestructura como código en *AWS (Amazon Web services)* y no *GCP (Google Cloud Platform)*, sin utilizar *docker* ni micro servicios e incluyendo *devops*.

Hablando de infraestructura como código, existe el desarrollo de un entorno que despliega un *clúster* utilizando *hadoop ansible* y *terraform*, reduciendo tiempo y esfuerzo de implementación para el almacenamiento y procesamiento de datos distribuidos. Utilizando *terraform* como aprovisionamiento y *ansible*, para la gestión de configuraciones. Todo automatizado sin intervención humana, configurando un *clúster* con *hadoop*, sobre un servidor en la nube, permitiendo conectarse a maquinas *Linux* y *Windows*, mediante conexión ssh.(Gupta et al. 2021)

Por el lado de *devops* se ha construido un *framework* con *continius delivery* para data *science*, utilizando algoritmos de reconocimiento fácil en un marco de prácticas *devops*.(Saxena et al. 2021) Desarrollando un canal de entrega para un sistema de aprendizaje automático mediante la extracción de información y reconocimiento a partir de datos, aplicando cuatro prácticas de *devops* en un sistema de aprendizaje automático, estas prácticas son el control de versiones, el servidor de modelos, la contenerización y *CI/CD*.

En la misma línea se ha utilizado herramientas para una solución de gestión de información educativa, utilizando herramientas como *Jenkins Kubernetes* y *git*. (Yang et al. 2020). Para mejorar la calidad de la gestión de los estudiantes, desarrollado un nuevo sistema de información educativa basado en el concepto de *devops* en in sistema de integración continúa basado en *Git*, despliegue continúo basado en *Jenkins* y *Kubernetes*, gestión de *logs* basado en *ELK* y Sistema de análisis de calidad de código basado en *SonarQube*.

7. APORTE DEL PROYECTO

Teniendo en cuenta la problemática expuesta anteriormente y el análisis realizado en el estado del arte, logramos observar que existe acercamientos con respecto a herramientas utilizadas en procesamiento de datos como lo es *spark*, *hadoop*, por el lado de *devops sonar*, *Jenkins*, *git*, por el lado visualización *jupyterhub* y *terraform* desplegando un clúster entre otros componentes o tecnologías pensados en la arquitectura, pero se carece de un desarrollo u entorno que configure y despliegue todas estas características mediante el modelo de despliegue de infraestructura como código.

El propósito y el aporte de este proyecto es lograr configurar e integrar un clúster para análisis, procesamiento y visualización de datos distribuidos mediante un despliegue de infraestructura como código enfocado a la práctica y experimentación, integrando la opción de *devops* en el ciclo de entrega continua, con una configuración *end to end* desde ceros y en lenguaje utilizando un lenguaje de configuración de alto nivel llamado *HCL* (HashiCorp Configuration Language).

Si bien algunos desarrollos de los componentes ya existen, trataremos de colocar un grado mayor de dificultad en la arquitectura propuesta, cuanto la configuración por ejemplo en la cantidad de nodos, *notebook* apuntando al *clúster* como interfaz de usuario, *sonarCloud* y *git*

actions como herramientas para el flujo de trabajo *devops*, realizando toda la configuración con *bash*, y *HCL* como ilustra la figura 1. Este desarrollo se disponibilizará en el repositorio de código *git*, para que los usuarios lo descarguen en sus equipos y puedan ejecutar el entorno con comandos *terraform*, luego de cumplir con los pre requisitos adjuntos en la sesión 2 y 3 del manual anexo en este documento.

Figura 1. Aporte de integración de componentes

8. FASE 1: ANÁLISIS

7.2 Arquitecturas de Referencia Verificadas:

Como vimos en el estado del arte, observamos que las arquitecturas big data son influyentes en temas de procesamientos de datos distribuidos, por lo que se toman como base para poder analizarlas y diseñar nuestra arquitectura.

7.2.1 Arquitecturas de procesamiento de datos distribuidos:

Las particularidades de los datos, involucran un desafío para las organizaciones, debido que se requieren gestionar datos con sus variantes características, de este modo se requiere arquitecturas específicas que posean componentes que almacenen, procesen, analicen y visualicen un gran volumen de variedad de datos.(Blazquez & Domenech, 2018)

7.2.1.1 Arquitectura Lambda:

La arquitectura Lambda mostrada en la Fig.2 tiene tres capas: *batch, speed y serving*. La **capa de lotes** gestiona los datos históricos y vuelve a calcular los resultados (aprendizaje automático). Itera a través de todos los datos y tiene una alta latencia.

La **capa de velocidad** procesa los datos entrantes y proporciona resultados en tiempo casi real. Los resultados no son tan precisos como los procesamientos por lotes(*batch*).

La **capa de servicio** obtiene los resultados de las otras dos capas y permite las consultas.

Aunque la arquitectura Lambda ofrece una gran flexibilidad, tiene un precio. La complejidad y la necesidad de mantener dos bases de código diferentes para el procesamiento de datos hace que sea difícil de implementar y soportar. (Zhelev & Rozeva, 2017)

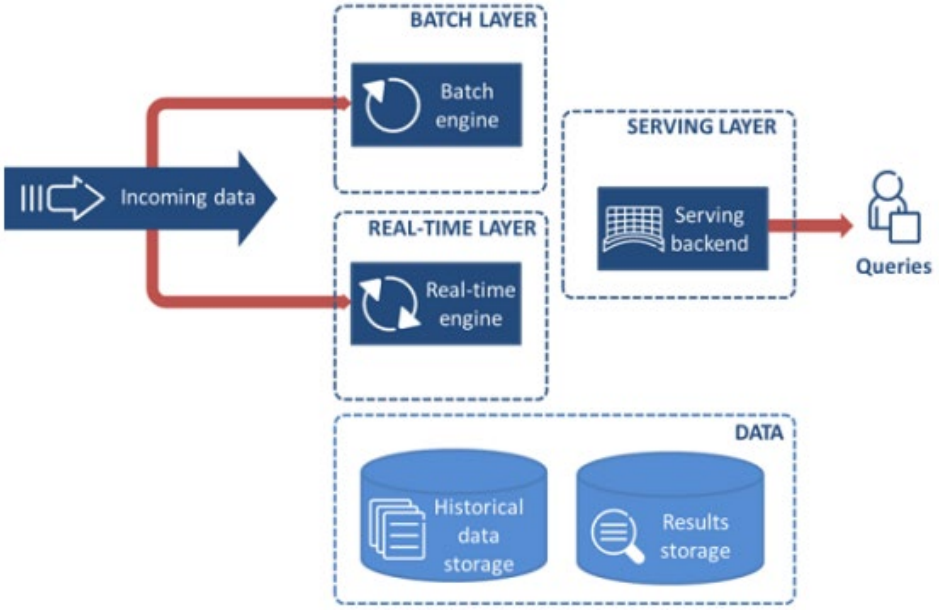


Figura 2 Arquitectura Lambda(Zhelev & Rozeva, 2017)

7.2.1.2 Arquitectura Kappa:

La arquitectura *Kappa*, mostrada en la Fig.3, trata de simplificar la arquitectura Lambda. Combina las capas *batch* y capas de velocidad en una sola y tiene sólo dos capas: procesamiento de flujos y servicio. La capa de procesamiento de flujos ejecuta

trabajos de procesamiento. Dependiendo de los datos que necesitemos procesar se ejecutan diferentes trabajos (datos en tiempo real, datos históricos). La capa de servicio se utiliza para consultar los resultados como en la arquitectura *Lambda*. *Kappa* cambia la flexibilidad por la simplicidad. (Zhelev & Rozeva, 2017)

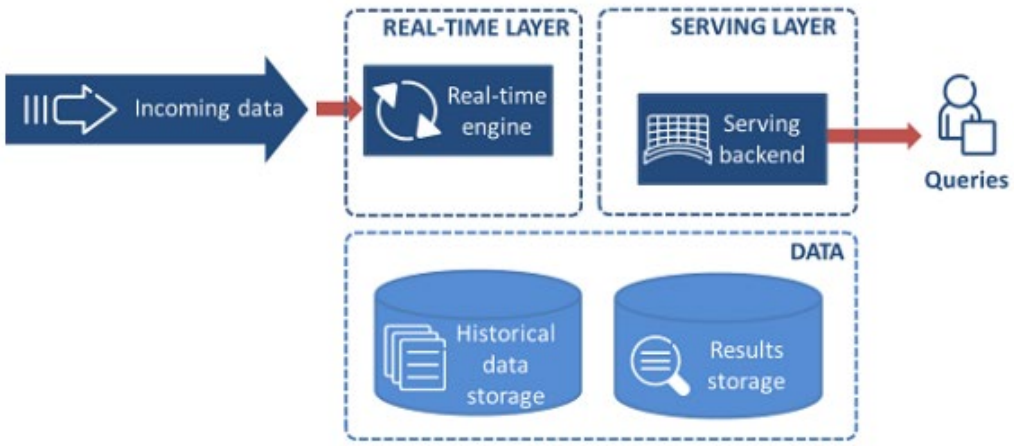


Figura 3Arquitectura Kappa(Zhelev & Rozeva, 2017)

7.2.1.3 Arquitectura Basada en Modelos:

Esta fue desarrollada para agilizar los procesos de analítica desde la preparación de los datos hasta la visualización. Se centra en la visualización, que tiene un papel clave en la analítica de *big data* para permitir a los usuarios comprender el problema, generar hipótesis y definir la solución, así como para dirigir el proceso de análisis cuando se trata de datos masivos, incompletos y datos incorrectos.(Golfarelli & Rizzi, 2020)

7.2.1.4 Arquitectura de Procesamiento de Big Data Propuesta Por Krishnan:

La arquitectura presentada por Krishnan (2010), consiste en cuatro etapas: recolección o recopilación, carga, transformación y extracción de datos.

7.2.1.5 Arquitectura de Big Data Propuesta por Bob Marcus:

El modelo de arquitectura por niveles fue propuesto por Bob Marcus con los siguiente componentes y circuito:

Fuentes de datos externas: El componente A, es parte de la arquitectura de datos que suministra las entradas de datos externos insumo para alimentar el sistema *big data*.

Secuencia y procesamiento *ETL*: Las tareas que se desarrolla en el componente B son filtrar y transformar los flujos de datos provenientes de los recursos externos.

Fundación altamente escalable: Con respecto al componente C, existen tres tipos de escalonamiento: El primero, a nivel de la infraestructura, El segundo se refiere a los almacenes de datos y el procesamiento buscando aprovechar las ventajas de los almacenes de datos distribuidos escalable.

Bases de datos operacionales y de Analíticas: En el componente D se proponen tres clases de bases de datos: Bases de datos analíticas. El análisis de bases de datos toma los datos procesados, Bases de datos operacionales manteniendo una excelente operación en lectura y escritura en general de forma eficiente, por ejemplo, las bases de datos *NoSQL* y finalmente en la memoria de datos ubicados en memorias cachés, que buscan minimizar escribir en disco los datos.

Analítica e interfaces de bases de datos:

El componente E consta de tres partes: Análisis de interfaces de procesos en lotes. Se refiere al tipo de interfaz usada para el procesamiento de datos que provienen en lotes o *Batch*, una interfaz interactiva permitiendo el almacenamiento en bases de datos escalables y una interfaz de análisis de datos en tiempo real para el manejo de eventos

Aplicaciones e interfaz de usuario: El componente F se refiere a las aplicaciones e interfaces de usuario, las cuales no deben ser algoritmos complejos, al usar grandes cantidades de datos distribuidos.

Servicios de apoyo Es el componente G. Estos servicios hacen referencia a los componentes necesarios para la implementación y gestión de sistemas robustos de Big Data, diseñar, implementar herramientas, toda la parte de seguridad y proceso.(L., Camargo et al., 2015)

7.2.1.6 Arquitectura Big Data Centro Hospitalario de Portugal:

La arquitectura de *Big Data*, en tiempo real, establecida e implementada por Gonçalves y otros en la unidad de cuidados intensito del Centro Hospitalario de Porto en Portugal, establece una solución utilizando herramientas de código abierto como lo es el clúster de apache *hadoop*, en un entorno online para la extracción, el gobierno, análisis, almacenamiento y procesamiento de la información, la arquitectura de big data tiene para los agentes adquisición de datos, inferencia, interfaz y gestión del conocimiento. Los agentes hacen que el sistema funcione mediante acciones automáticas, que realizan algunas tareas esenciales, como la recogida automática de datos y la actualización de los modelos predictivos, en tiempo real, sin necesidad de intervención humana.(Gonçalves et al., 2017)

7.2.1.7 Procesamiento de Big Data Basado en Contenedores Docker en Múltiples Nubes:

El sistema de procesamiento de *big data* basado en contenedores *Docker* en múltiples nubes, explora otra dimensión potencial de *Docker* para el análisis. Este sistema basado en contenedores es un marco económico y fácil de usar, obteniendo conocimientos y habilidades básicas de TI. Además, se puede desarrollar fácilmente en una sola máquina, en varias máquinas o en varias nubes.(Naik, 2017)

7.2.1.8 Aplicación de Big Data mediante un marco virtualización ligera en la Nube

Nos plantea una arquitectura implementada constantemente en los entornos empresariales de *big data* en la nube, como *MapReduce* y sus implementaciones como *Hadoop* y *Spark*, proporcionando una interfaz de programación de alto nivel productiva para el procesamiento de datos a gran escala y la analítica. *MapReduce* es un marco de *software* que permite a un clúster de ordenadores procesar un gran conjunto de datos en paralelo. Esto en un marco de trabajo con un marco de trabajo maestro en virtualizaciones basadas en hipervisores virtualizaciones y contenedorización como *Docker*.(Bhimani et al., 2017)

7.2.1.9 Despliegue de una arquitectura utilizando hadoop. ansible y terraform

La configuración del clúster consume mayor tiempo y esfuerzo de implementación para el almacenamiento y procesamiento de datos distribuidos. Una alternativa para automatizar este proceso se puede realizar mediante *terraform* como aprovisionamiento y *Ansible*, para la gestión de configuraciones. Todo automatizado sin intervención humana, configurando un *clúster* con *hadoop*, sobre un servidor en la nube, permitiendo conectarse a maquinas *Linux* y *Windows*, mediante conexión ssh. (Gupta et al., 2021)

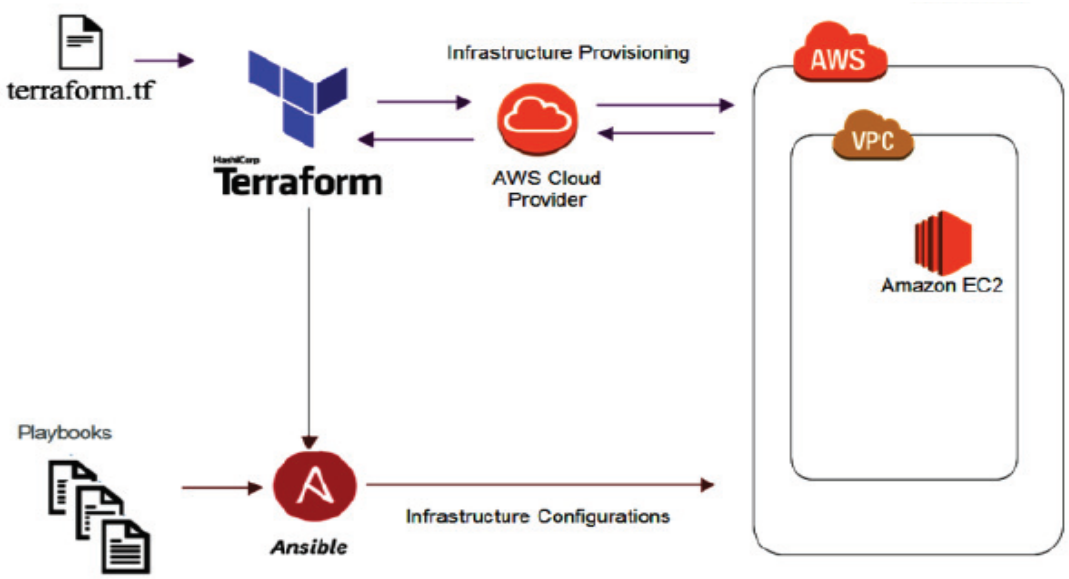


Figura 4. Automatización clúster con terraform y ansible. (Gupta et al. 2021)

7.2.2 Análisis de las arquitecturas de referencias vistas

En La revisión de las arquitecturas anteriores nos permitió identificar e integrar diferentes capas en el ciclo de vida de los datos, algunas herramientas de código libre disruptivas y que prevalecen en el tiempo y con ambientes de trabajo *on premise* como en la nube, y sin duda nos da idea de cómo diseñar la arquitectura para el entorno experimental que buscamos, además se observó en las mayorías de arquitecturas de referencia que la mayoría definen 5 capas:

- Fuentes de datos
- Carga de datos

- Transformación y almacenamiento
- Analítica
- Visualización

9. FASE 2: PLANEACIÓN Y DISEÑO

8.1 Diseño Arquitectura de Referencia Definida

Terraform como herramienta de software para la configuración y despliegue de recursos de infraestructura basados en los requisitos de la nube.

8.1.1 Fuentes de datos:

Esta capa permite acceder a los orígenes o fuentes de los datos recolectarlos. Las fuentes pueden ser: bases de datos internas o externas, documentos, imágenes u otros (Gonçalves et al. 2017).

8.1.2 Carga de Datos

En este punto se cargan los datos y se debe aplicar el concepto de metadatos, es decir, se deben describir las características de los datos que se cargan, como el nombre, la importancia y la relación con otros datos u objetos de la empresa. Uno de los objetivos de esta etapa es obtener una estructura homogénea que ayude a que los datos sean trazables y fáciles de acceder (L., Camargo et al. 2015)

8.1.3 Transformación y Almacenamiento

Durante esta capa los datos se transforman mediante la aplicación de reglas de negocio y el procesamiento de los datos y luego se almacenan.

8.1.4 Analítica:

Esta capa es responsable de procesar todo tipo de datos y realizar los análisis requeridos por la organización, el cual puede ser descriptivo, predictivo o prescriptivo

Visualización: En este punto se generan los resultados con informes de visualización y monitoreo de información. La generación de informes y tableros de control es una función

crítica en las arquitecturas de procesamiento de datos distribuidos, ya que esta capa debe permitir visualizar información útil.

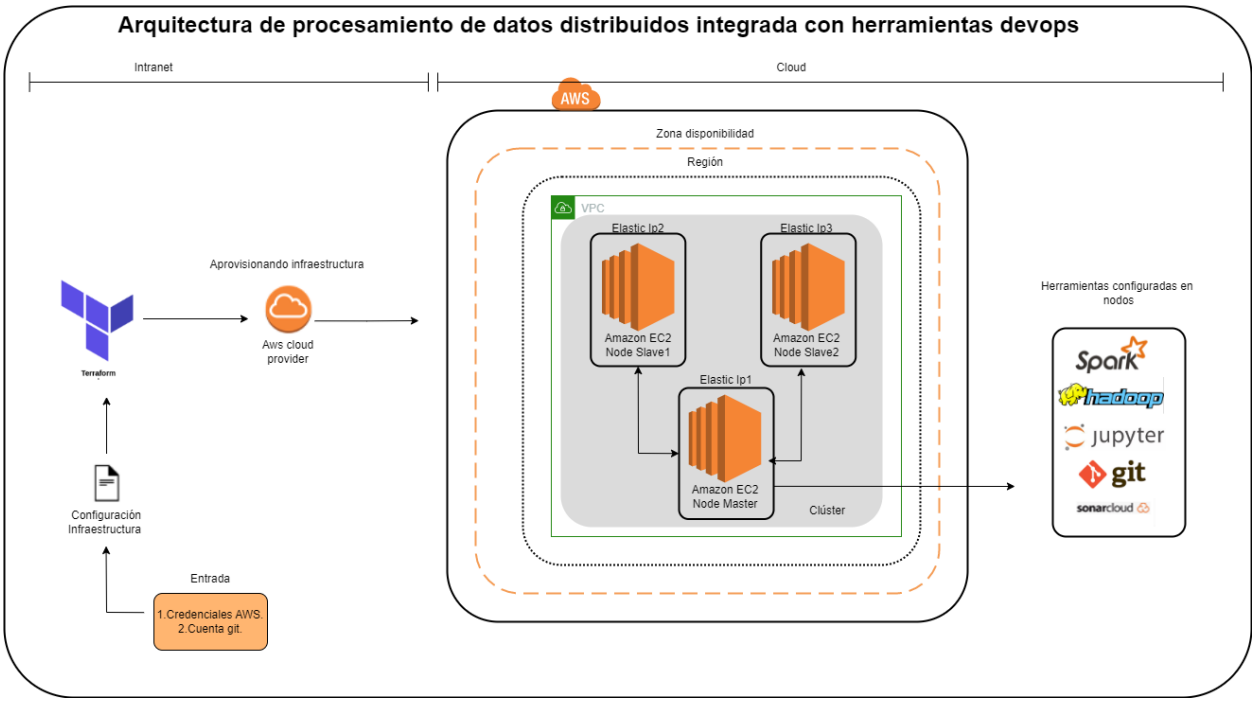


Figura 5. Arquitectura de procesamiento de datos distribuida definida integrada con herramientas Devops.

Esta arquitectura se desplegará mediante configuraciones *terraform*, aprovisionando la infraestructura en la nube de *AWS*, en el cual se dispone un clúster de procesamiento con un nodo maestro y 2 esclavos, bajo instancias *EC2 Linux micro* vacías, y toda la configuración se realizará desde *terraform*, adicional estos nodos se les configuro *spark, hadoop, notebook, git* y *sonar cloud* todo esto con las respectivas configuraciones de red. Como se puede apreciar en las figuras 5 y 6.

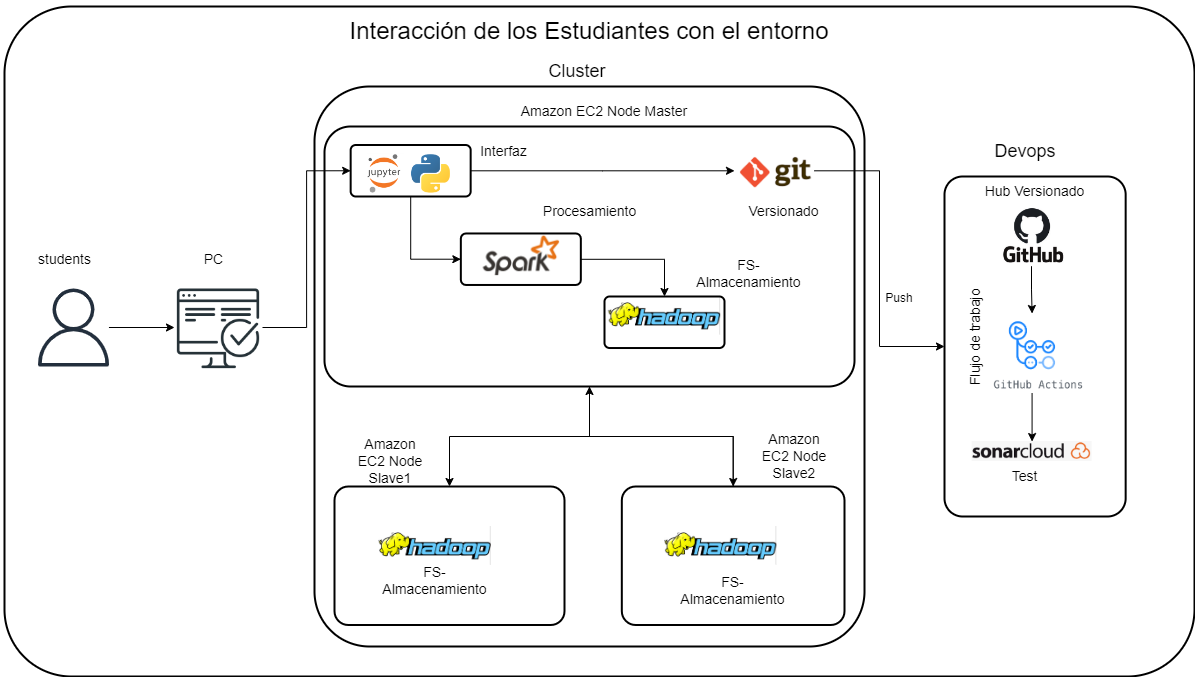


Figura 6. Interacción de los estudiantes con la arquitectura

8.1.5 Tecnologías o herramientas a utilizar

En la implementación del entorno se utilizarán diferentes tecnologías que trabajan entre sí, para lograr el objetivo de almacenar, procesar, analizar y visualizar grandes volúmenes de datos.

Las herramientas seleccionadas son las siguientes:

- Terraform
- Amazon Web Services (AWS)
- Apache Hadoop
- Apache Spark
- Jupyter Notebook
- GitHub
- Git Actions
- SonarQube(Cloud)

8.1.6 Descripción de Componentes en la Arquitectura:

Para la definiciones de recursos utilizados en la arquitectura ver manual anexo sección 1.2.

8.1.6.1 Terraform (Infraestructura como código):

Esta plataforma nos permitirá realizar la configuración y despliegue mediante instancias *Amazon EC2 linux*, el *clúster* que utilizaremos para nuestra arquitectura en la nube. (Anon n.d.)

9.1.6 Amazon Web Services (Proveedor servicios en la nube):

Se realizará el despliegue de la infraestructura en instancias Ec2, limpias sin ninguna configuración inicial, desarrollaremos toda la implantación del clúster desde cero.

9.1.7 Apache Spark (Procesamiento de datos):

Plataforma para el procesamiento de datos distribuidos en nuestro entorno.

8.1.6.4 Apache Hadoop (Almacenamiento):

Como proveedor del almacenamiento, utilizando su sistema de archivos. La forma en la que utilizaremos *spark* con *hadoop* será la siguiente:

Un gestor de clústeres se utilizará para adquirir recursos de clúster para ejecutar trabajos. El núcleo de *Spark* se ejecuta en el gestor de clústeres *Hadoop YARN* ubicado en instancias Amazon *EC2* y el gestor de clústeres integrado de *Spark* (es decir, independiente). El gestor de *clústeres* se encarga de compartir los recursos entre las aplicaciones de Spark. Por otro lado, Spark puede acceder a datos en *HDFS* mediante *Yarn*.

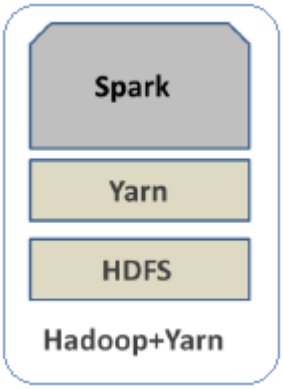


Figura 7. Despliegue de Spark en la arquitectura hadoop.

8.1.6.5 Jupyter Notebook (Visualización):

Herramienta de interacción con el usuario, mediante la misma se logrará interactuar con el entorno, obteniendo así una experticia muchos más visual y en la que se aprovisiona las configuraciones necesarias para poder realizar análisis, procesamiento y visualización de datos distribuidos.

9.1.8 Devops:

Esta opción se configura y parametriza, para que los estudiantes experimenten con este flujo y logren observar que siempre que se suban sus cambios en entornos colaborativos de desarrollo en empresas de tecnología, se desencadenaran un conjunto de acciones, en búsqueda de mejorar la calidad del código,

9.1.8.2 SonarQube(cloud):

Plataforma de análisis y gestión de la calidad del código.

9.1.8.3 Git:

Sistema de control de versiones *Git*, para el desarrollo de código. Cuando se utiliza *Git* para el desarrollo local, los archivos se convierten entre tres estados: modificado, en fase y confirmado. Estos estados están relacionados con el área de trabajo, el área de preparación y el repositorio local que *Git* mantiene localmente. En primer lugar, después del usuario extrae el código del repositorio local a través del comando *git checkout*, todas las modificaciones del código fuente se registran en el área de trabajo. En un periodo de tiempo, el código fuente actual modificado puede añadirse al área de almacenamiento temporal a través del comando *git add*, mientras que las modificaciones en el área de recorte no se añadirán. Finalmente, el usuario puede enviar los cambios en el área de almacenamiento temporal al repositorio local a través del comando *git commit*. Cada *commit* de *Git* registrará la estructura de árbol actual correspondiente al directorio de código fuente, el archivo actual actualizado y metadatos como los comentarios. (Yang et al. 2020)
Al realizar el *push Git* llamará ahora a la *URL* que desencadena la tarea de construcción y ejecución de *sonarQube Cloud* mediante la obtención el último código en la rama de distribución y utilizar las *GitAction* o flujos de trabajo de *git* para construir, compilar y finalmente probar los componentes de software.

10. FASE 3: CONFIGURACIÓN

10.1 Instalación ambiente experimental

Para revisar la descripción del entorno, los requerimientos de ejecución del entorno e instalaciones de componentes necesarios, al igual de la guía de cómo hacerlo; ver manual anexo sección 2.

Se realizará la configuración de los recursos en terraform en el siguiente orden:

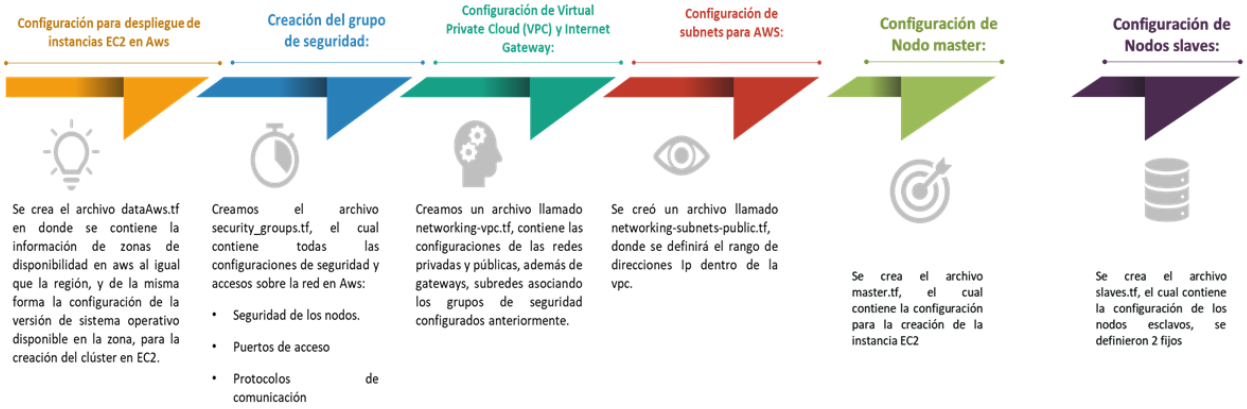


Figura 8. Configuración recursos en Aws

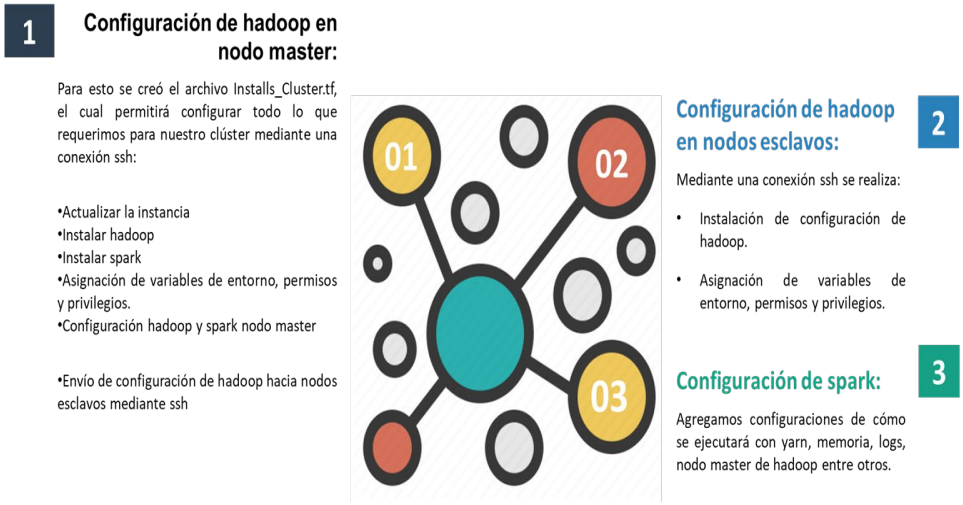


Figura 9.Configuración clúster

Figura 10. Configuración Jupyter Lab

Figura 11.Configuración Devops

10.1.1 Configuración recursos AWS:

10.1.1.1 Configuraciones credenciales:

Crearemos un archivo llamado credenciales.tf con la siguiente configuración, en donde debemos escribir access_key,secret_key con la calve descargada anteriormente, el *backend* de *terreform* quedara local en la ruta que le especifiquemos y asignar la región que le aparece disponible en *Aws* al momento de crear la cuenta:

```
# Backends variables and configuration
terraform {
  backend "local" {
    path = "D:/Entorno_BigData/terraform.tfstate"
  }
}

# Specify the provider and access details.
provider "aws" {
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
  region = var.aws_region
}
```

10.1.1.2 Configuración para despliegue de instancias EC2 en Aws:

Se crea el archivo dataAws.tf en donde se contiene la información de zonas de disponibilidad en *Aws* al igual que la región, y de la misma forma la configuración de la versión de sistema operativo disponible en la zona, para la creación del clúster en EC2.

```
data "aws_region" "current" {}
data "aws_availability_zones" "available" {
  state = "available"
}
data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name      = "name"
    values    = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-
server-*"]
  }

  filter {
    name      = "virtualization-type"
    values    = ["hvm"]
  }

  owners      = ["099720109477"] # Canonical
}
```

10.1.1.3 Creación del grupo de seguridad:

Creamos el archivo security_groups.tf, el cual contiene todas las configuraciones de seguridad y accesos sobre la red en Aws, este contempla el llamado a todos los archivos que se nombran a continuación.

```
# Our default security group for the master nodes.
resource "aws_security_group" "default_master" {
  depends_on = [aws_vpc.default]

  name = "${var.cluster_name}-master"
  description = "Spark Security Group: Master"
  vpc_id = aws_vpc.default.id

  # SSH access from anywhere.
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Outbound internet access.
  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  #Default security group that allows to ping the instance
  ingress {
    from_port        = -1
    to_port          = -1
    protocol         = "icmp"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

#sparkHistory server
  ingress {
    from_port = 18080
    to_port = 18080
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }


#notebook
  ingress {
    from_port = 4242
    to_port = 4242
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  #Security group for web that allows web traffic from internet
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
```

10.1.1.4 Configuración de Virtual Private Cloud (VPC) y Internet Gateway:

Se creó un archivo llamado networking-vpc.tf donde se definirá vpc y Gateway.

```
##
## Virtual Private Cloud (VPC).
##
resource "aws_vpc" "default" {
  cidr_block = var.vpc_cidr_block
  enable_dns_hostnames = true
  enable_dns_support = true

}

##
## Internet Gateway.
##
resource "aws_internet_gateway" "default" {
  depends_on = [aws_vpc.default]
  vpc_id = aws_vpc.default.id

}
```

10.1.1.5 Configuración de subnets para AWS:

Se creó un archivo llamado networking-subnets-public.tf, donde se definirá el rango de direcciones Ip dentro de la vpc.

```
# Public subnets.
resource "aws_subnet" "public" {
  depends_on = [aws_internet_gateway.default]

  vpc_id = aws_vpc.default.id
  cidr_block = var.public_subnet_cidr_block
  map_public_ip_on_launch = true

}

# Routing table for public subnet.
resource "aws_route_table" "public" {
  depends_on = [aws_internet_gateway.default]

  vpc_id = aws_vpc.default.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.default.id
  }
}

resource "aws_main_route_table_association" "main" {
    vpc_id = aws_vpc.default.id
    route_table_id = aws_route_table.public.id
}

# Associate the routing table to public subnet.
resource "aws_route_table_association" "public" {
  subnet_id = aws_subnet.public.id
  route_table_id = aws_route_table.public.id
}
```

10.1.1.6 Configuración de variables:

Se crea el archivo vars.tf el cual tiene algunas variables necesarias para el grupo de seguridad, para la conexión ssh y demás variables que se requieran.

```
variable "aws_access_key" {
  description = "AWS access key."
  default ="AKIAW365456Y7B45ANFF"
}

variable "aws secret key" {
  description = "AWS secret key."
  default ="AzJjxDsX13qGNOLUbWOyc/4grGSBS7"
}

variable "github_repository" {
  description = "Repository github"
  default ="https://github.com/charswat/prueba02.git"
}
variable "github token" {
  description = "personal access token"
  default ="ghp_jmXYJYsvwWyVMiMhnyCsa2kO"
}
variable "git_user_email" {
  description = "email."
  default ="charswat10@hotmail.com"
}

variable "git user name" {
  description = "user name"
  default ="charswat"
}

variable "sonar_projectKey" {
  description = "sonar project key"
  default ="charswat_bigData"
}

variable "sonar_organization" {
  description = "sonar organization"
  default ="charswat-1"
}

variable "ssh key name" {
  description = "The SSH key name to use for the instances."
  default = "proyecto"
}

variable "ssh_key_path" {
  description = "Path to the SSH private key file."
  default = "~/.ssh/proyecto.pem"
}


variable "vpc_cidr_block" {
  description = "The IPv4 address range that machines in the network
are assigned to, represented as a CIDR block."
  default = "172.31.0.0/16"
}

variable "public_subnet_cidr_block" {
  description = "The IPv4 address range that machines in the network
are assigned to, represented as a CIDR block."
  default = "172.31.0.0/20"
}
```

10.1.1.7 Configuración Nodo Master:

Se crea el archivo master.tf, el cual contiene la configuración para la creación de la instancia EC2, que se tomara como nodo master.

```
resource "aws_instance" "master_node" {
  depends_on = [aws_instance.slave_node]

  # communicate with the resource (instance).
  connection {
    # The default username for our AMI
    user = "ubuntu"

    # The path to your keyfile
    private_key = file(var.ssh_key_path)
    type = "ssh"
    host = self.public_ip
  }

  instance_type = var.aws_ec2_instance_type
  # Lookup the correct AMI based on the region
  # we specified.
  ami = data.aws_ami.ubuntu.id

  # The name of our SSH key-pair.
  key_name = var.ssh_key_name

  # Our security group to allow SSH access.
  security_groups             = [aws_security_group.default_master.id]
  subnet_id                   = aws_subnet.public.id
  vpc_security_group_ids      = [aws_security_group.default_master.id]
  associate_public_ip_address = true


  root_block_device {
    volume_type = "gp2"
    volume_size = 1000
  }
  tags = {
    Name    = "EC2 Master en ${aws_subnet.public.availability_zone}"

  }


  # This copies the dynamically generated list of slave node TCP/IP
addresses required by  hadoop and Spark

  provisioner "file" {
    source = "${path.module}/output/slaves"
    destination = "/tmp/slaves"
  }

  # Stage and execute the master node configuration script.
   provisioner "file" {
   source = "${path.module}/shells/key-rsa-master.sh"
   destination = "/tmp/key-rsa-master.sh"
  }
  # Stage and execute the master node configuration script.
  provisioner "file" {
    source = "${path.module}/shells/start-hadoop.sh"
    destination = "/tmp/start-hadoop.sh"
  }
  provisioner "file" {
    source = "${path.module}/shells/setup-spark-jupyter.sh"
    destination = "/tmp/setup-spark-jupyter.sh"
```

10.1.1.8 Configuración Nodos slaves:

Se crea el archivo **slaves.tf**, el cual contiene la configuración de los nodos esclavos, se definieron 2 fijos.

```
resource "aws_instance" "slave_node" {
  depends_on = [aws_security_group.default_slave]

  # The connection block tells our provisioner how to communicate with
the
  # resource (instance).
  connection {
    # The default username for our AMI
    user = "ubuntu"

    # The path to your keyfile
    private_key = file(var.ssh_key_path)

    type = "ssh"
    host = self.public_ip
    timeout = "1m"
  }

  instance_type = var.aws_ec2_instance_type

  # Lookup the correct AMI based on the region we specified.
  ami = data.aws_ami.ubuntu.id

  tags = {
    Name    = "EC2 Slave"

  }

  # The name of our SSH key-pair.
  key_name = var.ssh_key_name

  # Our Security group to allow SSH access.
  security_groups        = [aws_security_group.default_slave.id]
  vpc_security_group_ids = [aws_security_group.default_slave.id]
  subnet_id              = aws_subnet.public.id
  associate_public_ip_address = true

  root_block_device {
    volume_type = "gp2"
    volume_size = 1000
  }

  count = 2

  provisioner "local-exec" {
    command = "echo ${self.private_ip} >>
${path.module}/output/slaves"
  }

}
```

10.1.1.9 Configuración ips relacionadas con el clúster:

Creamos el archivo llamado **outputs.tf**, configurando los links a consultar por los estudiantes cuando el entorno termine el despliegue de toda su arquitectura, mediante esta configuración se informará las direcciones a las interfaces web de Yarn, hadoop, spark, SonarCloud ,Jupyterlab y la clave de acceso al mismo.

```
output "master_public_ip" {
  value = aws_instance.master_node.public_ip
}

output "yarn" {
  value = "http://${aws_instance.master_node.public_ip}:8080"
}
output "interface_hadoop_web" {
  value = "http://${aws_instance.master_node.public_ip}:9870"
}
output "interface_yarn_web" {
  value =
"http://${aws_instance.master_node.public_ip}:8088/cluster/nodes"
}
output "spark_History_server" {
  value = "http://${aws_instance.master_node.public_ip}:18080"
}
output "JupiterLab" {
  value = "http://${aws_instance.master_node.public_ip}:4242"
}
output "access_key_JupiterLab" {
  value = "1234"
}
```

10.1.1.10 Configuración de instalación de componentes en los nodos:

Para esto se creó el archivo Installs_Cluster.tf, el cual permitirá configurar todo lo que requerimos para nuestro clúster.

```
resource "null_resource" "master"  {


  connection {

    type     = "ssh"
    user     = "ubuntu"
    private_key = file(var.ssh_key_path)
    host     = aws_instance.master_node.public_ip

  }

  provisioner "remote-exec" {

    inline = [
      "sudo apt-get update",
      "sudo apt-get install git",
      "sudo apt install openssh-server openssh-client -y",
      "cd /home/ubuntu",
      "echo 'export HADOOP_HOME=/home/ubuntu/hadoop\nexport
PATH=$PATH:/home/ubuntu/hadoop/bin:/home/ubuntu/hadoop/sbin'| sudo tee
.bashrc",
      "echo '127.0.0.1
localhost\n${aws_instance.master_node.private_ip}
master\n${aws_instance.slave_node.0.private_ip}
slave01\n${aws_instance.slave_node.1.private_ip} slave02' | sudo tee
/etc/hosts",
      "echo
'PATH=/home/hadoop/hadoop/bin:/home/hadoop/hadoop/sbin:$PATH' | sudo
tee .profile",
      "echo config spark",
      "wget https://dlcdn.apache.org/spark/spark-3.3.0/spark-3.3.0-
bin-hadoop3.tgz",
      "tar -xvf spark-3.3.0-bin-hadoop3.tgz",
      "mv spark-3.3.0-bin-hadoop3 spark",
      "mv /home/ubuntu/spark/conf/spark-defaults.conf.template
/home/ubuntu/spark/conf/spark-defaults.conf",
      "wget https://archive.apache.org/dist/hadoop/common/hadoop-
3.3.0/hadoop-3.3.0.tar.gz",
      "sudo tar -xvzf hadoop-3.3.0.tar.gz",
      "mv hadoop-3.3.0 hadoop",
      "sudo rm -r hadoop.tar.gz",
      "sudo rm -r spark-3.3.0-bin-hadoop3.tgz",
      "sudo rm -r hadoop-3.3.0.tar.gz",
      "sudo usermod -a -G sudo ubuntu",
      "sudo chown ubuntu:root -R /home/ubuntu/hadoop/",
      "sudo chmod g+rwx -R /home/ubuntu/hadoop/",
      "sudo adduser ubuntu sudo",
      "sudo mkdir /home/ubuntu/tmpdata/",
      "sudo mkdir /home/ubuntu/hadoop/tmp/dir",
      "sudo mkdir /home/ubuntu/hadoop/hdfs",
      "sudo mkdir /home/ubuntu/hadoop/hdfs/datanode",
      "sudo mkdir /home/ubuntu/hadoop/hdfs/namenode",
      "sudo chown -R ubuntu /home/ubuntu/hadoop/hdfs/",
      "cd /home/ubuntu/hadoop",
      "sudo mkdir /ruta",
      "sudo chown -R ubuntu /home/ubuntu/hadoop/",
      "cd /home/ubuntu/hadoop/etc/hadoop",
      "echo ¡configuración archivos!",
      "sudo sed -i '/<configuration>/c
```

10.1.1.11 Despliegue de infraestructura mediante terraform:

Terraform aprovisionará infraestructura como código de las instancias EC2, Grupos de Seguridad, *VPC* en *AWS*, *Terraform* tiene cuatro comandos esenciales que nos permiten abordar un flujo de trabajo.

10.1.1.11.1 terraform init:

Permite inicializar *terraform* y descargar versiones tanto versiones como archivos del proveedor necesarios en este caso *Aws* como se ilustra en la 12.

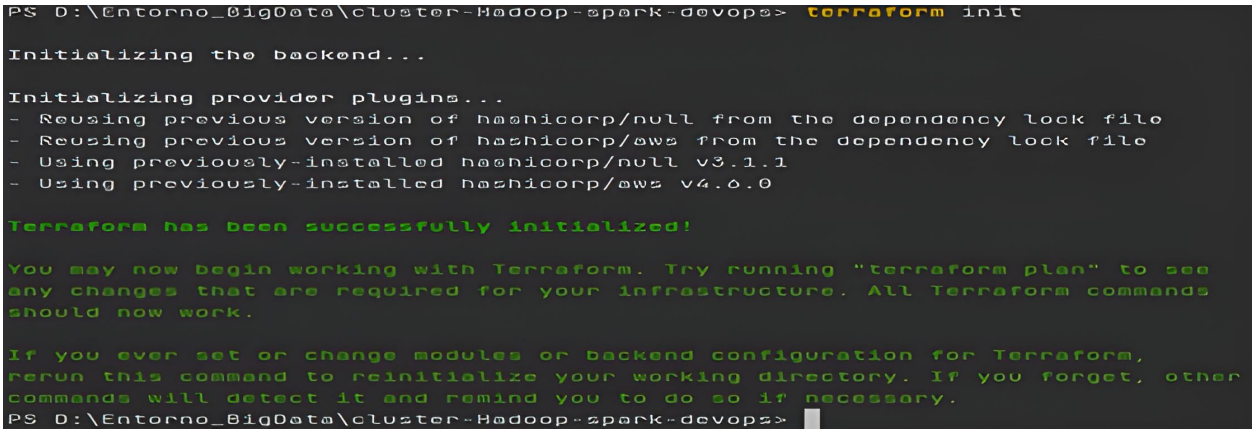


Figura 12. terraform init

10.1.1.11.2 terraform plan:

Este comando permite crear el plan de ejecución de la infraestructura programada en cada uno de los archivos de configuración previos, al final lista los componentes que creara como se observa en la Figura 13.

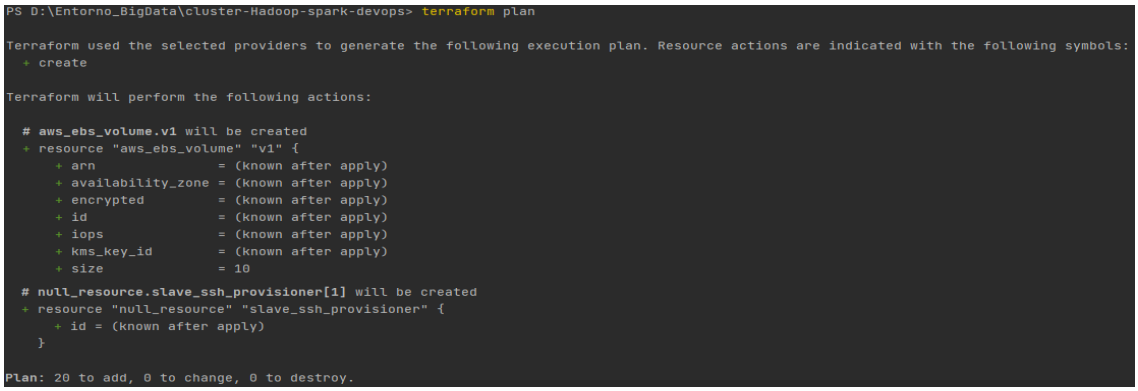


Figura 13. terraform plan

10.1.1.11.3 terraform apply:

Se utiliza para aplicar los cambios necesarios para alcanzar el estado deseado de la configuración, o el conjunto predefinido de acciones generado por un plan ejecución. Se utilizó el comando **apply -auto-approve**, que no pide confirmación de la acción.

10.1.1.11.4 terraform destroy:

El comando terraform destroy se utiliza para destruir la infraestructura gestionada por Terraform.

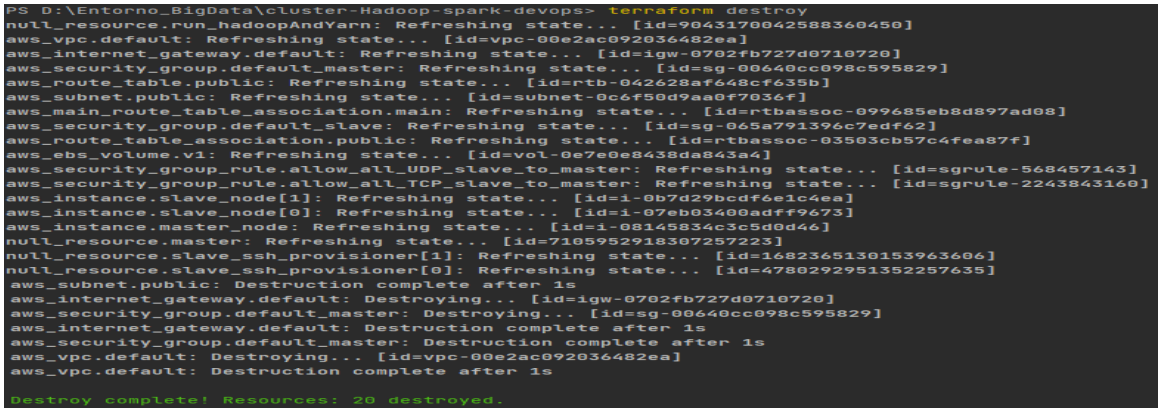


Figura 14. terraform destroy

10.1.1.11.5 Visualización Instancias creadas en AWS:

Luego de finalizar el comando anterior en la figura 15 se visualiza 3 instancias Ec2 corriendo, donde 1 va a ser el nodo maestro y 2 esclavos.

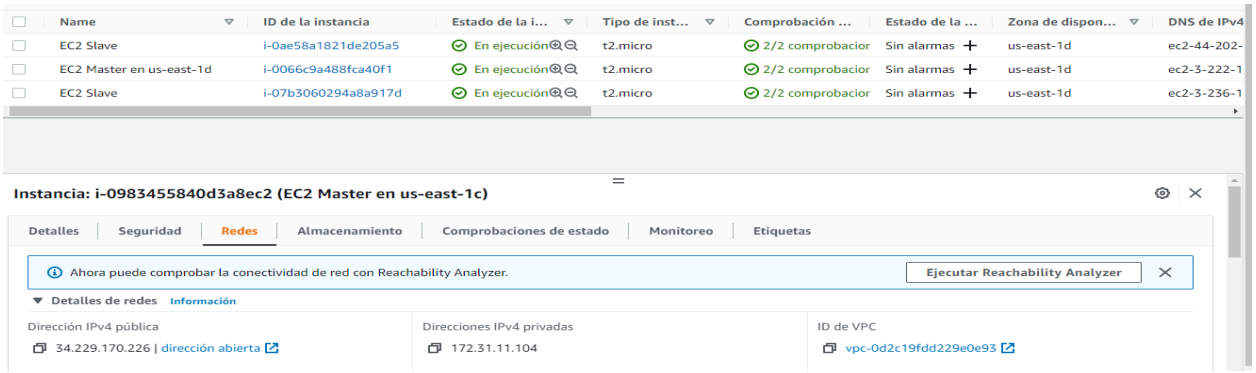


Figura 15.Ejecución Instancias Ec2 en AWS.

10.1.2 SSH a instancias EC2

10.1.2.1 local-exec y remote-exec:

Estos dos provisionadores son necesarios en Terraform, dado que carece de los plug-ins nativos necesarios. Esta es la solución para invocar dentro del aprovisionador a un local-exec. Requiere configurar la conexión con el host mediante ssh, usuario y private_key, ver un ejemplo Installs_Cluster.tf para más detalles.

```
resource "null_resource" "slave_ssh_provisioner" {
  depends_on = [aws_instance.master_node]

  count = var.cluster_size
  connection {
    user = "ubuntu"
    private_key = file(var.ssh_key_path)
    type = "ssh"
    timeout = "1m"
    host = element(aws_instance.slave_node.*.public_ip, count.index)
  }


  provisioner "remote-exec" {
    inline = [
      "sudo cat /home/ubuntu/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"
    ]
  }
```

Figura 16.Ejemplo remote-exec

10.1.3 Configuración de apache hadoop:

Para empezar, crearemos un clúster con 3 instancias, 1 maestro y 2 esclavos, mediante ips dinámicas creadas en el mediante el despliegue de los pasos anteriores. Resaltando que primero se crean los nodos esclavos y luego el nodo master.

Nos situamos en la carpeta /home/ubuntu/hadoop/etc/hadoop aquí se encuentran cinco archivos principales que fueron editados para que las tarjetas funcionaran en conjunto en modo clúster, declarando un nodo maestro y el resto esclavos, los archivos editados fueron **hdfs-site.xml, core-site.xml, yarn-site.xml, mapred-site.xml, workers, host.xml y hadoop-env.sh**, todos los archivos se configuraron de la misma manera.

10.1.3.1 Versión:

Instalaremos *Hadoop* en sus versiones binarias más recientes y estables en este caso Apache Hadoop versión 3.2.3.

10.1.3.2 Configuración nodo maestro:

10.1.3.2.1 Configuración ssh:

El nodo maestro utilizará una conexión SSH para conectarse a otros nodos con autenticación de par de claves. Esto permitirá que el nodo maestro administre activamente el clúster. Por lo que se realiza la conexión ssh mediante un remote-exec y se genera la creación del par de llaves, las cuales servirán para crear la carpeta ~/.ssh mediante un script bash llamado **key-rsa-master.sh.**

```
echo "Generating cluster's SSH key on master..."
if [ -f ~/.ssh/id_rsa ] ||
    (ssh-keygen -q -t rsa -N '' -f ~/.ssh/id_rsa &&
     cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys); then

  echo "Cluster SSH keys generated successfully."
fi
```

Luego de ello nos conectamos a los esclavos mediante ssh copiamos la llave privada y pública generadas en el nodo master y almacenadas en la ${path.module}/output/ y autorizamos a utilizarlas en cada nodo esclavo.

```
# Send the SSH public key up to the slave and add it to the
authorized hosts file.
provisioner "file" {
  source = "${path.module}/output/id_rsa.pub"
  destination = "/home/ubuntu/.ssh/id_rsa.pub"
}

# Send the SSH private key up to the slave and add it to the
authorized_hosts file.
provisioner "file" {
  source = "${path.module}/output/id_rsa"
  destination = "/home/ubuntu/.ssh/id_rsa"
}

provisioner "remote-exec" {
  inline = [
    "sudo cat /home/ubuntu/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"
  ]
}
```

10.1.3.2.2 Actualización complementos Linux:

```
sudo apt-get update
```

10.1.3.2.3 Instalación java:

```
sudo apt install openjdk-8-jdk -y
```

10.1.3.2.4 Instalación openssh-server:

```
sudo apt install openssh-server openssh-client -y
```

10.1.3.2.5 Instalación git:

```
sudo apt-get install git
```

10.1.3.2.6 Configuración hosts:

```
"echo '127.0.0.1 localhost\n${aws_instance.master_node.private_ip}
master\n${aws_instance.slave_node.0.private_ip}
slave01\n${aws_instance.slave_node.1.private_ip} slave02' | sudo tee
/etc/hosts",
```

10.1.3.2.7 Configuración. Profile

```
"echo 'PATH=/home/hadoop/hadoop/bin:/home/hadoop/hadoop/sbin:$PATH' |
sudo tee .profile",
```

10.1.3.2.8 Descargar y descomprimir binarios de Hadoop:

Para la descarga de los binarios ingresar a la página https://hadoop.apache.org/releases.html.

```
"wget https://archive.apache.org/dist/hadoop/common/hadoop-
3.3.0/hadoop-3.3.0.tar.gz",
```

10.1.3.2.9 Configuración de variable de entorno:

```
"echo 'export HADOOP_HOME=/home/ubuntu/hadoop\nexport
PATH=$PATH:/home/ubuntu/hadoop/bin:/home/ubuntu/hadoop/sbin'| sudo tee
.bashrc",
```

10.1.3.2.10 Configuración privilegios de usuario linux:

```
"sudo usermod -a -G sudo ubuntu",
"sudo chown ubuntu:root -R /home/ubuntu/hadoop/",
"sudo chmod g+rwx -R /home/ubuntu/hadoop/",
"sudo adduser ubuntu sudo"
```

10.1.3.2.11 Configuración core-site.xml:

Se establece la ubicación de NameNode en node-master en el puerto 9001.

```
"sudo sed -i '/<configuration>/c
<configuration><property><name>fs.default.name</name><value>hdfs://mas
ter:9001</value></property>' core-site.xml"
```

10.1.3.2.12 Configuración hdfs-site.xml:

Permite establecer la ruta del hdfs. La propiedad, dfs.replication, indica cuántas veces se replican los datos en el clúster. Se configuran 2 para que todos los datos se dupliquen en los dos nodos.

```
"sudo sed -i '/<configuration>/c
<configuration><property><name>dfs.replication</name><value>2</value><
/property><property><name>dfs.namenode.name.dir</name><value>file:/hom
e/ubuntu/hadoop/hdfs/namenode</value></property><property><name>dfs.da
tanode.data.dir</name><value>file:/home/ubuntu/hadoop/hdfs/datanode</v
alue></property>' hdfs-site.xml"
```

10.1.3.2.13 Configuración mapred-site.xml:

Se establece YARN como programador de trabajos, configurándolo como predeterminado para las operaciones de MapReduce, asignación de memoria a la máquina virtual de java, número

de tareas mínimas o máximas, número de núcleos, así como la dirección y puerto para el administrador de tareas y el historial de tareas entre otras.

```
"sudo sed -i '/<configuration>/c
<configuration><property><name>mapreduce.framework.name</name><value>y
arn</value></property><property><name>yarn.app.mapreduce.am.env</name>
<value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value></property><property><na
me>mapreduce.map.env</name><value>HADOOP_MAPRED_HOME=$HADOOP_HOME</val
ue></property><property><name>mapreduce.reduce.env</name><value>HADOOP
_MAPRED_HOME=$HADOOP_HOME</value></property><property><name>yarn.app.m
apreduce.am.resource.mb</name><value>2048</value></property><property>
<name>mapreduce.map.memory.mb</name><value>1024
</value></property><property><name>mapreduce.reduce.memory.mb</name><v
alue>1024</value></property>' mapred-site.xml"
```

10.1.3.2.14 Configuración yarn-site.xml:

Contiene las opciones de configuración para YARN.

```
"sudo sed -i '/<configuration>/c
<configuration><property><name>yarn.acl.enable</name><value>0</value><
/property><property><name>yarn.resourcemanager.hostname</name><value>$
{aws_instance.master_node.private_ip}</value></property><property><nam
e>yarn.nodemanager.aux-
services</name><value>mapreduce_shuffle</value></property><property><n
ame>yarn.nodemanager.resource.memory-
mb</name><value>9216</value></property><property><name>yarn.scheduler.
maximum-allocation-
mb</name><value>6144</value></property><property><name>yarn.scheduler.
minimum-allocation-
mb</name><value>1024</value></property><property><name>yarn.nodemanage
r.vmem-check-enabled</name><value>false</value></property>' yarn-
site.xml"
```

10.1.3.2.15 Configuración host.xml:

```
"echo '${aws_instance.master_node.private_ip}
master\n${aws_instance.slave_node.0.private_ip}
slave01\n${aws_instance.slave_node.1.private_ip} slave02' | sudo tee
host.xml"
```

10.1.3.2.16 Configuración Workers:

Los scripts de inicio utilizan el archivo para iniciar los demonios necesarios en todos los nodos.

```
"echo '${aws_instance.slave_node.0.private_ip}
slave01\n${aws_instance.slave_node.1.private_ip} slave02' | sudo tee
workers"
```

10.1.3.2.17 Configuración hadoop-env.sh:

Donde se establece jdk de java instado.

```
"echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre' | sudo
tee hadoop-env.sh"
```

10.1.3.2.18 Configuración directorios del sistema de archivos de hadoop:

Es necesario crear un directorio en el cual Hadoop manejará los archivos temporales y directorios que se van a crear, por lo tanto, fue necesario asignar un directorio en común para el sistema de archivos de Hadoop (HDFS) entre los nodos esclavos, así como también un directorio específico en el nodo maestro.

Estos directorios fueron declarados en el archivo hdfs-site.xml, por lo que es necesario crear los directorios.

```
"sudo mkdir /home/ubuntu/hadoop/hdfs",
"sudo mkdir /home/ubuntu/hadoop/hdfs/datanode",
"sudo mkdir /home/ubuntu/hadoop/hdfs/namenode"
```

10.1.3.2.19 Configuración inicial ssh a nodos esclavos:

Esta configuración permite que no se tenga que enviar confirmación de conexión ssh hacia los nodos esclavos y sobre el mismo nodo master, toma como base lo que se parametrizo en la hoja 64 sección 7.1.6.2.6 Configuración hosts: y automáticamente establece la conexión mediante las llaves configuradas en secciones pasadas.

```
"ssh -o 'StrictHostKeyChecking no' ubuntu@slave01 exit ",
"ssh -o 'StrictHostKeyChecking no' ubuntu@slave02 exit ",
"ssh -o 'StrictHostKeyChecking no' ubuntu@master  exit",
"ssh -o 'StrictHostKeyChecking no' ubuntu@localhost  exit",
"ssh -o 'StrictHostKeyChecking no' ubuntu@0.0.0.0  exit"
```

10.1.3.3 Configuración nodos esclavos:

10.1.3.3.1 Configuración archivos hadoop:

Se realiza mediante él envió de la carpeta /home/Ubuntu/hadoop comprimida en un tar.gz y se envía por medio de conexión **SCP**, que es una herramienta de Linux que permite el copiado de archivos cifrados.

```
"ssh -o 'StrictHostKeyChecking no' ubuntu@slave01 exit ",
"ssh -o 'StrictHostKeyChecking no' ubuntu@slave02 exit ",
"ssh -o 'StrictHostKeyChecking no' ubuntu@master  exit",
"ssh -o 'StrictHostKeyChecking no' ubuntu@localhost  exit",
"ssh -o 'StrictHostKeyChecking no' ubuntu@0.0.0.0  exit"
```

10.1.3.3.2 Actualización complementos Linux:

```
sudo apt-get update
```

10.1.3.3.3 Instalación java:

```
sudo apt install openjdk-8-jdk -y
```

10.1.3.3.4 Instalación openssh-server:

```
sudo apt install openssh-server openssh-client -y
```

10.1.3.3.5 Configuración hosts:

```
"echo '127.0.0.1 localhost\n${aws_instance.master_node.private_ip}
master\n${aws_instance.slave_node.0.private_ip}
slave01\n${aws_instance.slave_node.1.private_ip} slave02' | sudo tee
/etc/hosts"
```

10.1.3.3.6 Configuración privilegios de usuario linux:

```
"sudo usermod -a -G sudo ubuntu",
  "sudo chown ubuntu:root -R /home/ubuntu/hadoop/",
  "sudo chmod g+rwx -R /home/ubuntu/hadoop/",
  "sudo adduser ubuntu sudo"
]
```

10.1.4 Configuración apache spark:

10.1.4.1 Recordemos el funcionamiento de spark en nuestra arquitectura:

Spark es un sistema informático de clúster de propósito general. Puede implementar y ejecutar aplicaciones paralelas en clústeres que van desde un solo nodo hasta miles de nodos distribuidos. Spark se diseñó originalmente para ejecutar aplicaciones Scala, pero también es compatible con Java, Python y R.

Recuerden que vamos a ejecutar spark como administrador de clúster independiente y trabajando con yarn como clúster dedicado.

Los trabajos de Spark se pueden ejecutar en YARN en dos modos: modo de clúster y modo de cliente. Comprender la diferencia entre los dos modos es importante para elegir una configuración de asignación de memoria adecuada y enviar trabajos como se esperaba.

Un trabajo de Spark consta de dos partes: ejecutores de Spark que ejecutan las tareas reales y un controlador de Spark que programa los ejecutores.

10.1.4.1.1 Modo clúster:

todo se ejecuta dentro del clúster. Puede iniciar un trabajo desde su computadora portátil y el trabajo continuará ejecutándose incluso si cierra su computadora. En este modo, el Spark Driver está encapsulado dentro del maestro de aplicaciones YARN.

10.1.4.1.2 Modo cliente:

El controlador Spark se ejecuta en un cliente, como su computadora portátil. Si el cliente se apaga, el trabajo falla. Spark Executors aún se ejecuta en el clúster y, para programar todo, se crea una pequeña aplicación principal de YARN.

El modo de cliente es adecuado para trabajos interactivos, pero las aplicaciones fallarán si el cliente se detiene. Para trabajos de ejecución prolongada, el modo de clúster es más adecuado.

10.1.4.2 Versión:

Se realiza la versión consecuente con la descargada de hadoop, es decir apache spark 3.2.2. Lo que es clave para que se comuniquen de la mejor

manera.

10.1.4.3 Descarga e instalación archivos binarios de Spark:

Descargamos los binarios de la página https://spark.apache.org/downloads.html, esta configuración al igual que la de hadoop, se utiliza el archivo **Installs_Cluster.tf** para la descarga de los binarios y la Shell **setup-spark-jupyter.sh** para realizar las respectivas configuraciones del mismo.

```
"wget https://dlcdn.apache.org/spark/spark-3.3.0/spark-3.3.0-bin-
hadoop3.tgz",
"tar -xvf spark-3.3.0-bin-hadoop3.tgz",
"mv spark-3.3.0-bin-hadoop3 spark"
```

10.1.4.4 Agregando variables de entorno:

10.1.4.4.1 Configuración. bashrc:

```
sudo sed -i '/sbin/c export PATH=$PATH:~/.local/bin\nexport
PATH=$PATH:/home/ubuntu/hadoop/bin:/home/ubuntu/hadoop/sbin\nSPARK_PAT
H=/home/ubuntu/spark' ~/.bashrc
```

10.1.4.4.2 Configuración. profile:

```
echo  export PATH=/home/ubuntu/spark/bin:$PATH
HADOOP_CONF_DIR=/home/ubuntu/hadoop/etc/hadoop export
HADOOP_CONF_DIR=/home/ubuntu/hadoop/etc/hadoop export
SPARK_HOME=/home/ubuntu/spark export
LD_LIBRARY_PATH=/home/ubuntu/hadoop/lib/native:$LD_LIBRARY_PATH | sudo
tee .profile
```

10.1.4.5 Integrar Spark con YARN

Hay que modificar primero el nombre del archivo de configuración de la plantilla predeterminada de Spark.

```
"mv /home/ubuntu/spark/conf/spark-defaults.conf.template
/home/ubuntu/spark/conf/spark-defaults.conf",
```

Luego de ello modificamos el mismo archivo, agregando configuraciones como que se ejecutara con yarn, memoria, logs, nodo master de hadoop entre otros.

```
sudo sed -i '/# Example:/c # Example:\nspark.master
yarn\nspark.yarn.am.memory  512m\nspark.executor.memory
1024m\nspark.eventLog.enabled true\nspark.eventLog.dir
hdfs://master:9001/spark-logs\nspark.history.provider
org.apache.spark.deploy.history.FsHistoryProvider\nspark.history.fs.lo
gDirectory hdfs://master:9001/spark-
logs\nspark.history.fs.update.interval 10s\nspark.history.ui.port
18080' /home/ubuntu/spark/conf/spark-defaults.conf
```

10.1.5 Configuración de visualización JupiterLab:

Vamos a instalar el servidor Jupyterlab, este tiene un entorno de desarrollo basado en la web para Jupyter Notebook que también admite complementos para agregar funcionalidad.

Jupyterlab está disponible a través de pip . Sin embargo, instalar Jupyterlab implica instalar una larga lista de dependencias de Python. Para evitar conflictos de dependencia, instalaremos Jupyterlab en un entorno virtual llamado **venv.**

10.1.5.1 Instalación entorno virtual:

Primero, usamos apt-get para instalar venv para Python 3

```
sudo apt-get install python3-venv
```

Luego creamos un directorio para el entorno virtual llamado dl-venv. Esto se hace usando el comando **venv**

```
python3 -m venv dl-venv
```

El comando venv crea un directorio llamado dl-venv en /home/Ubuntu
Este directorio contiene un intérprete de Python, bibliotecas y scripts que están aislados del resto del sistema. Cuando activa el entorno virtual e instala una nueva biblioteca de Python, la biblioteca se instala aquí en lugar del directorio de Python del sistema. Cuando desactiva el entorno virtual, estas bibliotecas ya no son accesibles para el intérprete de Python del sistema.

Luego activemos el entorno virtual dl-venv . venv proporciona un script para hacerlo

Este proceso se realiza mediante el archivo **setup-spark-jupyter.sh**, el cual contiene la siguiente configuración:

10.1.5.2 Instalación Python:

```
sudo apt install python3-pip -y
```

10.1.5.3 Instalación JupiterLab:

```
pip3 install jupyterlab
```

10.1.5.4 Configuraciones variables. profile:

Se adicionan variables de entorno a las agregadas anteriormente en la configuración de hadoop.

```
echo  export PATH=/home/ubuntu/spark/bin:$PATH
HADOOP_CONF_DIR=/home/ubuntu/hadoop/etc/hadoop export
HADOOP_CONF_DIR=/home/ubuntu/hadoop/etc/hadoop export
SPARK_HOME=/home/ubuntu/spark export
LD_LIBRARY_PATH=/home/ubuntu/hadoop/lib/native:$LD_LIBRARY_PATH | sudo
tee .profile
```

10.1.5.5 Configuraciones variables .bashrc:

Se adicionan variables de entorno a las agregadas anteriormente en la configuración de *hadoop*.

```
sudo sed -i '/sbin/c export PATH=$PATH:~/.local/bin\nexport
PATH=$PATH:/home/ubuntu/hadoop/bin:/home/ubuntu/hadoop/sbin\nSPARK_PATH=~/spark\ne
xport PYSPARK_DRIVER_PYTHON=\"jupyter\"\nexport
PYSPARK_DRIVER_PYTHON_OPTS=\"lab\"\n$SPARK_PATH/bin/pyspark --master
yarn\nPYSPARK_PYTHON=python3' ~/.bashrc
export PATH=$PATH:~/.local/bin
```

10.1.5.6 Comandos para guardar cambios de variables:

Estos comandos aplicaran los cambios realizados en, bashrc y .profile, estos dos archivos básicamente son los que lee el sistema Linux para la aplicación de variables de entorno.

```
source ~/.bashrc
source ~/.profile
```

10.1.5.7 Modificación Configuración archivos jupyterLab:

Se agregan los parámetros que nos permitirán establece una configuración especifica al jupyter, configuraciones como clave acceso, cambio puerto ejecución y otros.

```
sudo sed -i "/#c.TerminalManager.cull_interval = 300/c
c.NotebookApp.ip = "\"0.0.0.0\""\nc.NotebookApp.open_browser =
False\nc.NotebookApp.token="\"1234\""\nc.NotebookApp.allow_root=True\n
c.NotebookApp.port =4242" .jupyter/jupyter_notebook_config.py
```

```
(dl-venv) ubuntu@ip-172-31-0-71:~/dl-venv/share/jupyter/kernels/pyspark3$ echo '{
>       "argv": [
>         "python",
>         "-m",
>         "ipykernel_launcher",
>         "-f",
>         "{connection_file}"
>       ],
>       "display_name": "Pyspark 3",
>       "language": "python",
>       "env": {
>         "PYSPARK_PYTHON": "/usr/bin/python3",
>         "SPARK_HOME": "/opt/spark3/",
>         "SPARK_OPTS": "--master yarn --conf spark.ui.port=0",
>         "PYTHONPATH": "/opt/spark3/python/lib/py4j-0.10.9-src.zip:/opt/spark3/python/"
>       }
> }' | sudo tee /home/ubuntu/dl-venv/share/jupyter/kernels/pyspark3/kernel.json
```

Figura 17. Configuración archivos jupyterLab

```
(dl-venv) ubuntu@ip-172-31-0-71:~/dl-venv$ jupyter kernelspec install /home/ubuntu/dl-venv/share/jupyter/kernels/pyspark3 --user
```

Figura 18. Código instalación pyspark

```
(dl-venv) ubuntu@ip-172-31-0-71:~/dl-venv$ jupyter kernelspec list
Available kernels:
  pyspark3    /home/ubuntu/.local/share/jupyter/kernels/pyspark3
  python3     /home/ubuntu/.local/share/jupyter/kernels/python3
(dl-venv) ubuntu@ip-172-31-0-71:~/dl-venv$
```

Figura 19. lista de kernel jupyter

10.1.6 Configuración flujo entrega continua:

Para verificar las configuraciones en general de *git* ver el manual anexo en la sección 3.2.

10.1.6.1 Automatización de clonado del repositorio:

Para realizar este proceso de manera automática, hemos realizado la creación de una Shell llamada **"continuous-delivery.sh"** y se ejecutará en el nodo master siguiente manera:

```
#!/usr/bin/env bash

source ~/.profile

if [ -z "$EMAIL" ] || [ -z "$NAME" ]|| [ -z "$TOKEN" ]||[ -z
"$REPOSITORY" ]; then
    echo "[error] parámetros must be defined." >&2
    exit 1
fi
 #config git
 cd /home/ubuntu
 git config --global user.name "$NAME"
 git config --global user.email "$EMAIL"
 git config --local --unset credential.helper
 git config --global --unset credential.helper "cache --timeout=86400"
 git config --global user.password $TOKEN
 #clonar repositorio
 git clone $REPOSITORY Versionado
 cd Versionado
 git remote set-url origin $REPOSITORY
 sudo mkdir -p .github/workflows
 sudo mv /home/ubuntu/build.yml
/home/ubuntu/Versionado/.github/workflows/build.yml
 sudo mv /home/ubuntu/sonar-project.properties
/home/ubuntu/Versionado/sonar-project.properties
```

10.1.6.2 Aprovisionando archivo continuous delivery Github:

Para realizar este proceso de manera automática agregaremos al nodo master el archivo continuous-delivery:

```
provisioner "file" {
  source = "${path.module}/shells/continuous-delivery.sh"
  destination = "/tmp/continuous-delivery.sh"
}
provisioner "remote-exec" {
  inline = [
    "sudo echo 'export EMAIL=${var.git_user_email}' >>
$HOME/.profile",
    "sudo echo 'export NAME=${var.git_user_name}' >> $HOME/.profile",
    "sudo echo 'export TOKEN=${var.github_token}' >> $HOME/.profile",
    "sudo echo 'export REPOSITORY=${var.github_repository}' >>
$HOME/.profile",
    "bash /tmp/continuous-delivery.sh"

  ]
}
```

10.1.6.3 Configuración variable GitHub

Ahora bien, previamente para que el proceso se realizó de manera dinámica se han agregado a las variables de nuestro entorno, específicamente en el archivo vars.tf, las siguientes variables con el fin de tenerlas por defecto.

```
variable "aws_access_key" {
  description = "AWS access key."
  default ="AKIAW3DXTZ8UUB45ANFF"
}

variable "aws_secret_key" {
  description = "AWS secret key."
  default ="tyreyuyuyruihd/4grGSBS7"
}

variable "github_repository" {
  description = "Repository github"
  default ="https://github.com/charswat/prueba02.git"
}
variable "github_token" {
  description = "personal access token"
  default ="ghp_jmXYJYsvwWRWTRyC9yHQo3WPrk20sa2kO"
}
variable "git_user_email" {
  description = "email."
  default ="charswat1@hotmail.com"
}

variable "git_user_name" {
  description = "user name"
  default ="charswat"
}
```

10.1.6.4 Configuración SonarCloud y Creación variable sonar_token en github:

Para verificar la configuraciónes en general de *SonarCloud* ver el manual anexo en la sección 3.3.

Esto nos arroja unos parámetros de configuración, los cuales debemos relacionar en nuestro desarrollo en 2 archivos, el primero le daremos el nombre de "**SonarQube.yml**", el cual siempre será configurado automáticamente, el segundo archivo se llamará "**sonar-project.properties**".

SonarQube.yml:

```yaml
name: Build
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  sonarcloud:
    name: SonarCloud
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
        with:
          fetch-depth: 0  # Shallow clones should be disabled for a
better relevancy of analysis
      - name: SonarCloud Scan
        uses: SonarSource/sonarcloud-github-action@master
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Needed to get
PR information, if any
          SONAR_T
```

sonar-project.properties:

```
sonar.organization=charswat_prueba022
sonar.projectKey=charswat-1
# This is the name and version displayed in the SonarCloud UI.
#sonar.projectVersion=1.0

# Path is relative to the sonar-project.properties file. Replace "\"
by "/" on Windows.
#sonar.sources=.

# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
```

10.1.6.4.1 Aprovisionamiento de archivos SonarCloud en el entorno:

Estas líneas de código son configuradas en nodo master.

```
provisioner "file" {
  source = "${path.module}/shells/sonar-project.properties"
  destination = "/home/ubuntu/sonar-project.properties"
}
provisioner "file" {
  source = "${path.module}/shells/build.yml"
  destination = "/home/ubuntu/build.yml"
}
```

10.1.6.4.2 Configuración de variables para parámetros:

Se realiza la creación de unas variables para recibir la información automáticamente desde los parámetros en entrada de la ejecución del ambiente.

```
variable "sonar_projectKey" {
  description = "sonar project key"
  default ="charswat_bigData"
}

variable "sonar_organization" {
  description = "sonar organization"
  default ="charswat-1"
}
```

11. FASE 4: EJECUCIÓN Y DIAGNÓSTICO

11.1 Ejecución entorno:

11.1.1 Implementación del clúster:

11.1.1.1 Formateo desde el nodo master:

Se le aplica el comando hdfs namenode –format, este comando lo aplicamos en un script de nombre *start-hadoop.sh*, el cual también utilizamos para aplicar las variables de entorno colocadas en el archivo *.bashrc*, como las configuradas en él .profile.

```
#!/usr/bin/env bash
#Execute config   .profile where is vars
source ~/.profile
#Execute config   .bashrc where is vars
source ~/.bashrc
#format nameNode
 hdfs namenode -format -force
```

Una vez terminado el proceso nos arrojó en la terminal el siguiente resultado lo cual significó que todo se realizó correctamente:

```
2022-05-10 02:51:30,082 INFO util.GSet: VM type       = 64-bit
2022-05-10 02:51:30,082 INFO util.GSet: 0.029999999329447746% max memory 235.9 MB = 72.5 KB
2022-05-10 02:51:30,084 INFO util.GSet: capacity      = 2^13 = 8192 entries
Data exists in Storage Directory root= /home/ubuntu/hadoop/hdfs/namenode; location= null. Formatting anyway.
2022-05-10 02:51:30,149 INFO namenode.FSImage: Allocated new BlockPoolId: BP-2077462529-172.31.13.211-1652151090130
2022-05-10 02:51:30,150 INFO common.Storage: Will remove files: [/home/ubuntu/hadoop/hdfs/namenode/current/fsimage_0000000000000000003.md5, /home/ubuntu/hadoop/hdf
000000000004-0000000000000000005, /home/ubuntu/hadoop/hdfs/namenode/current/fsimage_0000000000000000003, /home/ubuntu/hadoop/hdfs/namenode/current/VERSION, /home/u
nt/seen_txid, /home/ubuntu/hadoop/hdfs/namenode/current/edits_inprogress_0000000000000000006, /home/ubuntu/hadoop/hdfs/namenode/current/edits_0000000000000000001-0
/hadoop/hdfs/namenode/current/fsimage_0000000000000000005.md5, /home/ubuntu/hadoop/hdfs/namenode/current/fsimage_0000000000000000005]
2022-05-10 02:51:30,195 INFO common.Storage: Storage directory /home/ubuntu/hadoop/hdfs/namenode has been successfully formatted.
2022-05-10 02:51:30,269 INFO namenode.FSImageFormatProtobuf: Saving image file /home/ubuntu/hadoop/hdfs/namenode/current/fsimage.ckpt_0000000000000000000 using no
2022-05-10 02:51:30,400 INFO namenode.FSImageFormatProtobuf: Image file /home/ubuntu/hadoop/hdfs/namenode/current/fsimage.ckpt_0000000000000000000 of size 401 byte
2022-05-10 02:51:30,418 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2022-05-10 02:51:30,440 INFO namenode.FSNamesystem: Stopping services started for active state
2022-05-10 02:51:30,441 INFO namenode.FSNamesystem: Stopping services started for standby state
2022-05-10 02:51:30,447 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2022-05-10 02:51:30,447 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at master/172.31.13.211
```

Figura 20.hdfs namenode -format

11.1.1.2 Iniciar sistema de archivos:

En una terminal del nodo maestro se ejecutó la siguiente instrucción para que el nodo maestro inicie los nodos esclavos y ponga en marcha el clúster:

start-dfs.sh

Esto iniciará NameNode y SecondaryNameNode en node-master, y DataNode en nodo1 y nodo2 , de acuerdo con la configuración del archivo workers.

La verificación de que los demonios estén corriendo se realiza bajo el comando jps, el cual deberá arrojar una salida así en el nodo master:

```
21922 Jps
21603 NameNode
21787 SecondaryNameNode
```

En los nodos esclavos:

```
19728 DataNode
19819 Jps
```

También puede usar automáticamente la interfaz de usuario web tanto de sistemas de archivos de hadoop como yarn, para el cual se configuro el archivo **outputs.tf**, el cual contiene las ips de interés del clúster. Apunte su navegador a http://node-master-IP:9870, donde node-master-IP es la dirección IP de su nodo-master, y obtendrá una consola de monitoreo fácil de usar.

11.1.1.3 Iniciar yarn:

Importante verificar que los procesos del dfs estén corriendo mediante el comando jps.

start-yarn.sh

se debería ver un ResourceManager en node-master y un NodeManager en node1 y node2.

Figura 21.Sistema de archivos de hadoop corriendo

```
15713 ResourceManager
8722 NameNode
8996 SecondaryNameNode
16022 Jps
```

Nodos Esclavos:

```
10822 NodeManager
10311 DataNode
11069 Jps
```

También puede usar automáticamente la interfaz de usuario web.

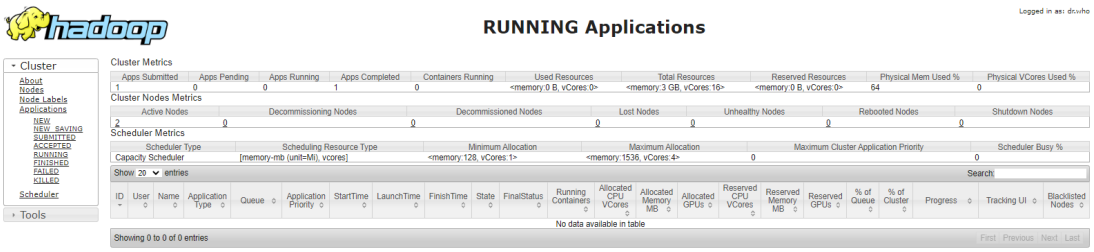


Figura 22.Visualización de Aplicaciones corriendo en MapReduce.

11.1.2 Prueba clúster:

11.1.2.1 Interacción con Hdfs:

Para lograr tener una comunicación e interacción con el sistema de archivos de hadoop (HDFS), se utiliza el comando hdfs dfs. Primero, crearemos manualmente un directorio de inicio donde todos los demás comandos utilizarán una ruta relativa a esta ruta de inicio predeterminada, ejecutar el siguiente comando:

```
hdfs dfs -mkdir -p /prueba/hadoop
```

11.1.2.2 Descarga de información para analizar:

Para ello realizaremos la descarga de un set de datos para realizar conteo de palabras, por lo que podemos acceder a una página de libros gratis del proyecto Gutenberg para poderlos analizar https://www.gutenberg.org, en el caso nuestro hemos elegido el libro de Frankenstein el cual descargamos de la siguiente url: wget -O frankenstein.txt https://www.gutenberg.org/files/84/84-0.txt,

11.1.2.3 Guardar el libro al almacenamiento del clúster:

En este caso creamos una carpeta en el sistema de almacenamiento de hadoop(hdfs) llamada book y luego guardamos en ella el libro descargado, para realizar este proceso debemos realizar el siguiente comando:

```
hdfs dfs -mkdir /book
hdfs dfs -put frankenstein.txt /book
hdfs dfs -ls /book
```

```
ubuntu@ip-172-31-0-71:~$ hdfs dfs -mkdir book
mkdir: `hdfs://localhost:9001/user/ubuntu': No such file or directory
ubuntu@ip-172-31-0-71:~$ hdfs dfs -mkdir /book
ubuntu@ip-172-31-0-71:~$ hdfs dfs -put frankenstein.txt /book
ubuntu@ip-172-31-0-71:~$ hdfs dfs -ls /book
Found 1 items
-rw-r--r--   2 ubuntu supergroup     448821 2022-09-10 15:19 /book/frankenstein.txt
```

Figura 23.Almacenamiento en hdfs

11.1.2.4 Ejecución conteo de palabras desde Yarn:

Finalmente lanzamos una ejecución del conteo de palabras del libro anteriormente relacionado mediante yarn bajo el siguiente comando:

```
yarn jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.3.jar wordcount "/book/frankenstein.txt" output
```

```
ubuntu@ip-172-31-0-71:~$ yarn jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.3.jar wordcount "/book/frankenstein.txt" output
2022-09-10 15:43:04,583 INFO client.RMProxy: Connecting to ResourceManager at master/172.31.0.71:8032
2022-09-10 15:43:05,402 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/ubuntu/.staging/job_1662824520567_0001
2022-09-10 15:43:06,449 INFO input.FileInputFormat: Total input files to process : 1
2022-09-10 15:43:06,639 INFO mapreduce.JobSubmitter: number of splits:1
2022-09-10 15:43:07,120 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1662824520567_0001
2022-09-10 15:43:07,125 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-09-10 15:43:07,537 INFO conf.Configuration: resource-types.xml not found
2022-09-10 15:43:07,538 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-09-10 15:43:08,175 INFO impl.YarnClientImpl: Submitted application application_1662824520567_0001
2022-09-10 15:43:08,240 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1662824520567_0001/
2022-09-10 15:43:08,241 INFO mapreduce.Job: Running job: job_1662824520567_0001
```

Figura 24.Ejecución wordcount

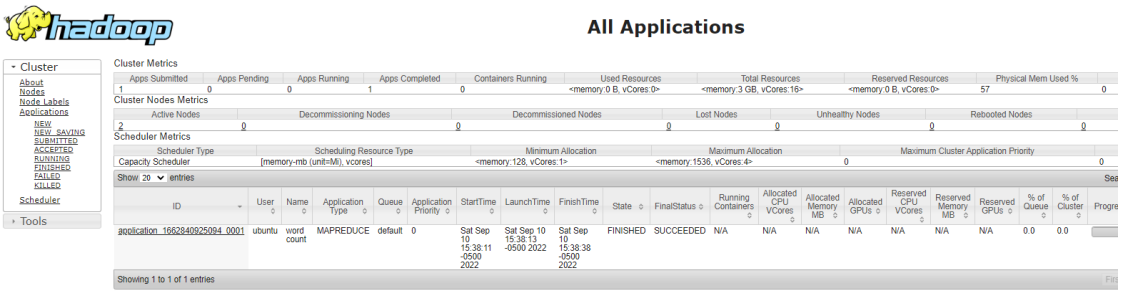


Figura 25.Flujo de trabajo en Yarn web

Una vez finalizado el trabajo, se puede obtener el resultado consultando HDFS con hdfs dfs -ls output. En caso de éxito, la salida se parecerá a:

```
(dl-venv) ubuntu@ip-172-31-2-9:~$ hdfs dfs -ls output
Found 2 items
-rw-r--r--   2 ubuntu supergroup          0 2022-09-10 20:38 output/_SUCCESS
```

```
-rw-r--r--   2 ubuntu supergroup     129593 2022-09-10 20:38 output/part-r-00000
Imprime el resultado con:
hdfs dfs -cat output/part-r-00000* | less
```

```
"Defects,"      1
"Information    1
"Plain  2
"Project        5
"Right  1
#84]    1
$5,000) 1
'AS-IS',        1
("the   1
($1     1
(801)   1
(Godwin)        3
(I      3
(July   1
(May    1
(a)     1
(although       2
(and    1
(any    1
(as     1
(b)     1
(c)     1
(does   1
(foolish        1
(for    2
(if     1
(inexperience   1
(or     3
(sight  1
(so     1
```

11.1.3 Especificaciones de clúster creado:

1.89 TB, 2 nodos esclavos,

11.1.4 Ejecución entorno jupyterLab:

Para desplegar el entorno se debe ejecutar el siguiente parámetro:

```
jupyter lab
```

Esta ejecución arrojara la dirección privada por donde nos podemos conectar, tener en cuenta que la configuración del clúster fue realizada previamente en la definición de las variables de entornos.

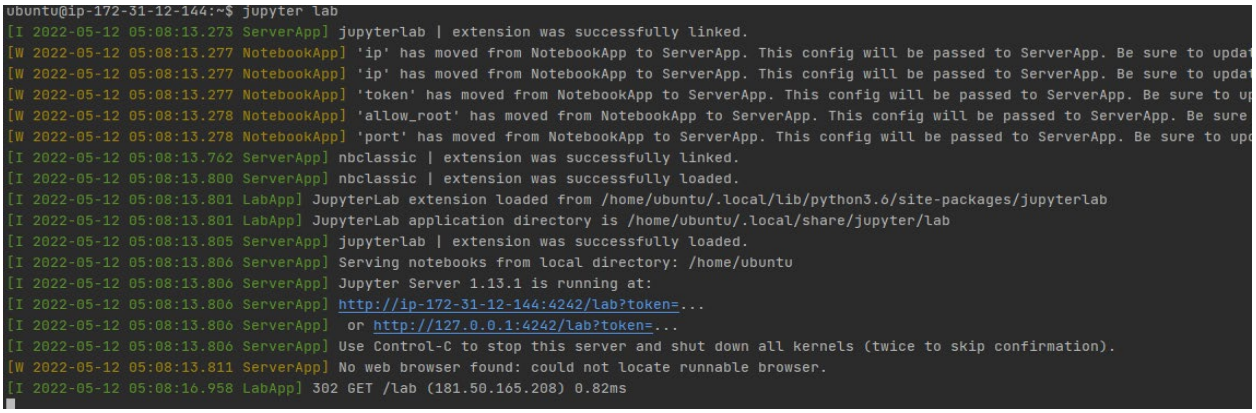


Figura 26.Ejecución JupyterLab

Hay que ingresar a la ip publica y puerto definido 4242(ippublica:4242) del nodo master para lograr interactuar con el notebook, mediante la **contraseña "1234"** y este nos permitirá realizar almacenamiento, análisis, procesamiento y visualización mediante Python y el uso de la terminal para manipular el sistema distribuido de hadoop previamente configurado.

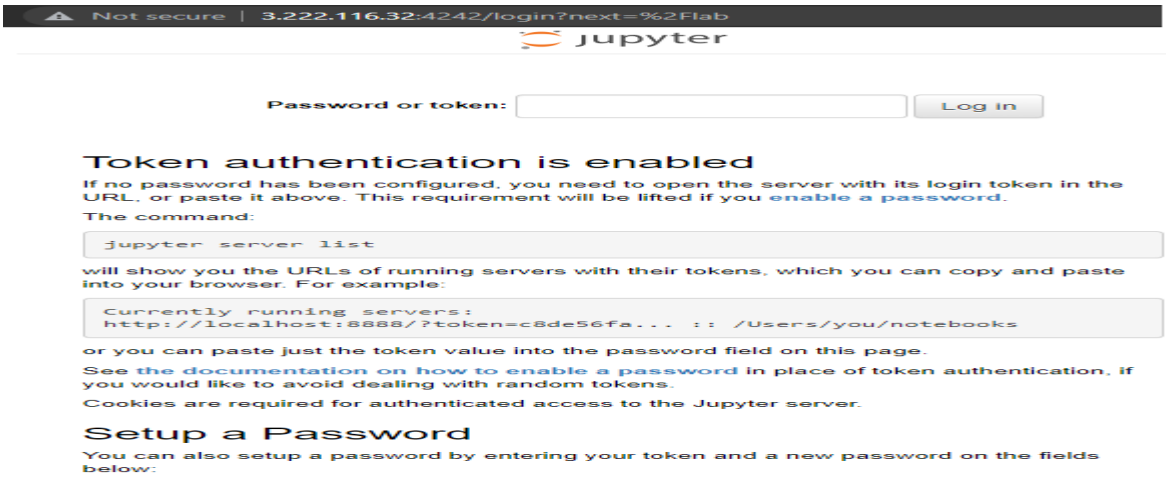


Figura 27.Despliegue de JupyterLab

Luego de ingresar la contraseña tendremos la ventana de la figura 28, la cual estará lista para empezar a desarrollar y practicar.

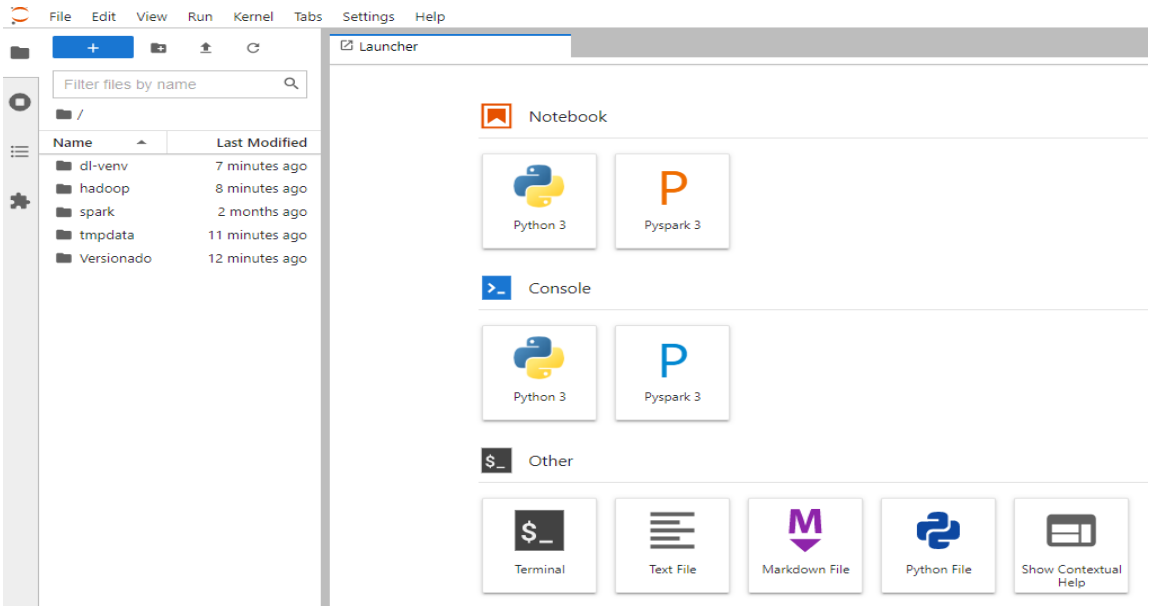


Figura 28.Jupyter listo para desarrollar

12. FASE 5: PRUEBAS Y ANÁLISIS DE RESULTADOS

12.1 Fase de Recolección de datos

Para probar la infraestructura fue necesario buscar un caso de uso con la suficiente complejidad para realizar una prueba de todos los elementos de la misma, por lo que se eligió el COVID 19. Sin duda alguna la pandemia del COVID 19 trajo un cambio en todos los aspectos de la vida de la población humana, por ejemplo, cambio de las modalidades de trabajo donde empresas se pasaron a entornos digitales generando mayor flujo de información e interacción con la nube, abriendo la posibilidad de trabajo remoto y laborar con empresas del exterior. Lo cual creo una demande de perfiles con habilidades en la nube y manipulación de datos distribuidos.

Por lo mencionado anteriormente y por querer conocer el impacto que dejo la pandemia en la ciudad de Bogotá, que elegimos analizar los datos abiertos del repositorio nacional de Colombia. Partiendo de que los datos desglosados de alta calidad, accesibles, fiables, oportunos, abiertos y fidedignos son fundamentales para generar información valiosa para la toma de decisiones en tiempo real.

12.1.1 Descripción caso del caso a implementar:

Se plantea realizar un análisis de los casos confirmados del COVID 19 en la ciudad de Bogotá, donde se logre identificar información más detallada por localidad de la que nos ofrece la página de la que nos ofrece la página https://coronaviruscolombia.gov.co/Covid19/index.html. Limitando

la muestra a 30.000 registros. Se toma este set de datos por ser un episodio que recordamos todos y se hace interesante que afectación tuvo en el país.

12.1.2 Identificación de los orígenes de los datos:

Para esta fase se utilizarán datos abiertos del gobierno nacional (Bogotá D.C), en este caso utilizando una muestra de la información de casos confirmados de COVID 19 de la ciudad de Bogotá. Al ingresar al a pagina https://datosabiertos.bogota.gov.co/dataset/numero-de-casos-confirmados-por-el-laboratorio-de-covid-19-bogota-d-c.

12.1.3 Obtención de los datos:

Luego de identificar la fuente de datos a analizar en el punto anterior se ingresa a la página y se procede a realizar la descarga como se visualiza la imagen siguiente, la cual permite descargar 2 archivos, el primero para la descarga de la muestra correspondiente a todos los sets de datos a analizar de casos confirmados de COVID 19 y el segundo la información de cada uno de los metadatos correspondiente a la descripción de las columnas e información que contendrá la muestra.

Figura 29.Descarga de set de datos COVID 19 Bogotá

Se han generado los siguientes archivos osb_enftransm-covid-19_18082022.csv y metadato_casos-confirmados-covid19.csv

12.2 Implementación caso de uso:

12.2.1 Almacenamiento de los datos:

Este proceso se realizará a través del cargue de información que nos permite realizar jupyter, en este caso solo cargaremos el archivo “osb_enftransm-covid-19_18082022.csv”.

Para subir el set de datos a analizar deberemos dar clic en la fecha hacia arriba como lo muestra la Figura 26, Esta acción abrirá la opción de ubicar en el pc el archivo descargado, dependiendo de lo que pese el archivo, este tardará algunos segundos.

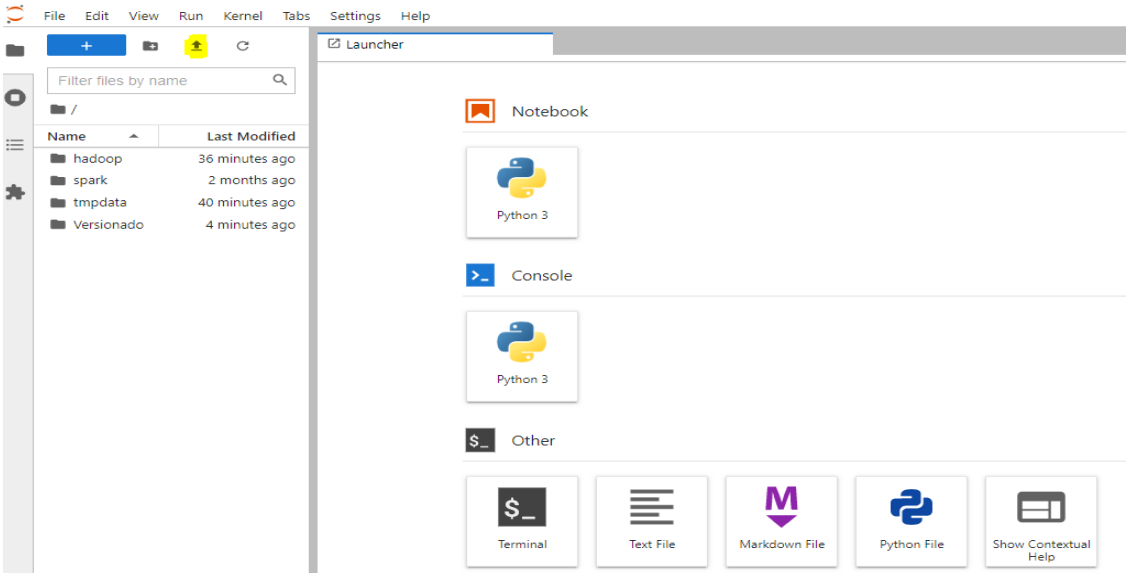


Figura 30.Almacenamiento de información en notebook

En esta sección realizaremos un análisis relativamente sencillo de como funcionaria el entorno con un set de datos, y algunas sentencias de con spark con Python. Teniendo en cuenta este lenguaje como base de toda implementación en nuestro entorno de desarrollo. Luego de cargados los datos, procedemos a abrir el notebook, que nos permitirá realizar el desarrollo del caso de uso.

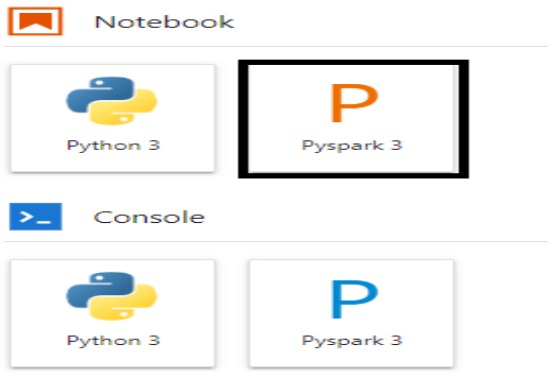


Figura 31.Notebook Python para desarrollo

12.2.2 Importación librerías spark:

Realizaremos la importación de librerías de spark y crearemos nuestra aplicación con spark mediante los siguientes parámetros:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
spark = SparkSession. \
    builder. \
    appName('implementacion-caso-de-uso'). \
    master('yarn'). \
    getOrCreate()
```

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
```

```
spark = SparkSession. \
    builder. \
    appName('implementacion-caso-de-uso'). \
    master('yarn'). \
    getOrCreate()
```

Figura 32.Import librerías spark en Python

12.2.3 Lectura archivo mediante spark:

Ya situados en el notebook listo para implementar el caso de uso y con el archivo subido al *jupyter* realizaremos la lectura mediante el siguiente comando y realizaremos un conteo de registros, asignando a la variable data todo el *dataframe* que se crear al momento de la lectura del insumo.

```
data =spark.read.format("csv").option("header",
"true").load("file:///home/ubuntu/Versionado/osb_enftransm-covid-19_18082023.csv")
```

Vale la pena resaltar que la carpeta versionado es la que contendrá el flujo de integración continua.

Figura 33.Lectura muestra y conteo de registros en ella para analizar.

Imprimiremos el *schema* que tiene el *dataframe* con el comando data.printSchema el cual arroja lo siguiente:

```
<bound method DataFrame.printSchema of DataFrame[CASO: string,
FECHA_DE_INICIO_DE_SINTOMAS: string, FECHA_DIAGNOSTICO: string, CIUDAD: string,
LOCALIDAD_ASIS: string, EDAD: string, UNI_MED: string, SEXO: string,
FUENTE_O_TIPO_DE_CONTAGIO: string, UBICACION: string, ESTADO: string]>
```

Con la sentencia data.show(10,false), mostraremos 10 registros así:

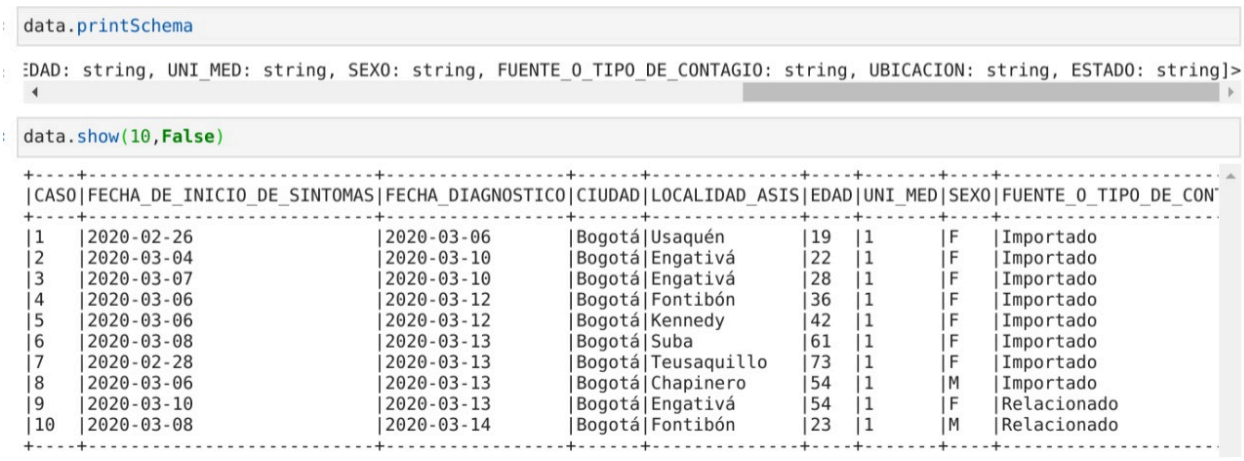


Figura 34.Imprimiendo schema y realizando un show a 10 registros.

12.2.4 Caso 1. Fallecidos y recuperados por localidad por COVID 19:

```
#fallecidos y recuperados por localidad a causa del COVID 19
fallecidos = data.groupby("LOCALIDAD_ASIS","ESTADO").count().orderBy("LOCALIDAD_ASIS")
fallecidos.show(truncate=False)
```

```
#fallecidos y recuperados por localidad a causa del covid 19
fallecidos = data.groupby("LOCALIDAD_ASIS","ESTADO").count().orderBy("LOCALIDAD_ASIS")
```

```
fallecidos.show(truncate=False)
```

```
+--------------+-------------------------------------+-----+
|LOCALIDAD_ASIS|ESTADO                               |count|
+--------------+-------------------------------------+-----+
|Antonio Nariño|Fallecido                            |31   |
|Antonio Nariño|Fallecido (No aplica No causa Directa)|8    |
|Antonio Nariño|Recuperado                           |504  |
|Barrios Unidos|Recuperado                           |396  |
|Barrios Unidos|Fallecido                            |28   |
|Barrios Unidos|Fallecido (No aplica No causa Directa)|5    |
|Bosa          |Recuperado                           |3182 |
|Bosa          |Fallecido                            |83   |
|Bosa          |Fallecido (No aplica No causa Directa)|10   |
|Chapinero     |Fallecido (No aplica No causa Directa)|2    |
|Chapinero     |Fallecido                            |16   |
|Chapinero     |Recuperado                           |583  |
|Ciudad Bolívar|Fallecido (No aplica No causa Directa)|16   |
|Ciudad Bolívar|Recuperado                           |2316 |
|Ciudad Bolívar|Fallecido                            |91   |
|Engativá      |Recuperado                           |2046 |
|Engativá      |Fallecido (No aplica No causa Directa)|17   |
|Engativá      |Fallecido                            |73   |
|Fontibón      |Recuperado                           |1302 |
|Fontibón      |Fallecido (No aplica No causa Directa)|14   |
+--------------+-------------------------------------+-----+
```

Figura 35.fallecidos y recuperados por localidad a causa del COVID 19

12.2.5 Caso 2. Fuente tipo de contagio por edad:

```
#fuente tipo de contagio por edad
tipoContagio = data.groupby("FUENTE_O_TIPO_DE_CONTAGIO",F.col("EDAD").cast("int").alias("EDAD")).count().orderBy("EDAD")
tipoContagio.show(truncate=False)
```

```
#fuente tipo de contagio por edad
tipoContagio = data.groupby("FUENTE_O_TIPO_DE_CONTAGIO",F.col("EDAD").cast("int").alias("EDAD")).count().orderBy("E
```

```
tipoContagio.show(truncate=False)
```

```
+-------------------------+----+-----+
|FUENTE_O_TIPO_DE_CONTAGIO|EDAD|count|
+-------------------------+----+-----+
|En estudio               |1   |67   |
|Desconocido              |1   |8    |
|Relacionado              |1   |114  |
|Desconocido              |2   |8    |
|Relacionado              |2   |114  |
|En estudio               |2   |53   |
|En estudio               |3   |31   |
|Relacionado              |3   |121  |
|Desconocido              |3   |3    |
|Relacionado              |4   |92   |
|Desconocido              |4   |3    |
|En estudio               |4   |37   |
|Importado                |5   |1    |
|Desconocido              |5   |3    |
|Relacionado              |5   |113  |
|En estudio               |5   |36   |
|Relacionado              |6   |103  |
|Desconocido              |6   |3    |
|En estudio               |6   |43   |
|Importado                |6   |1    |
+-------------------------+----+-----+
only showing top 20 rows
```

Figura 36.Fuente tipo de contagio por edad.

12.2.6 Caso 3. Fallecidos y recuperados por edad:

```
#fallecidos y recuperados por edad
estado = data.groupby("ESTADO",F.col("EDAD").cast("int").alias("EDAD")).count().orderBy("EDAD")
estado.show(truncate=False)
```

```
#fallecidos y recuperados por edad
estado = data.groupby("ESTADO",F.col("EDAD").cast("int").alias("EDAD")).count().orderBy("EDAD")
```

```
estado.show(truncate=False)
```

```
+-------------------------------------+----+-----+
|ESTADO                               |EDAD|count|
+-------------------------------------+----+-----+
|Recuperado                           |1   |185  |
|Fallecido (No aplica No causa Directa)|1   |4    |
|Recuperado                           |2   |174  |
|Fallecido (No aplica No causa Directa)|2   |1    |
|Fallecido (No aplica No causa Directa)|3   |1    |
|Recuperado                           |3   |154  |
|Recuperado                           |4   |132  |
|Recuperado                           |5   |151  |
|Fallecido (No aplica No causa Directa)|5   |2    |
|Recuperado                           |6   |150  |
|Recuperado                           |7   |170  |
|Recuperado                           |8   |170  |
|Fallecido (No aplica No causa Directa)|9   |1    |
|Recuperado                           |9   |171  |
|Recuperado                           |10  |203  |
|Recuperado                           |11  |221  |
|Recuperado                           |12  |211  |
|Recuperado                           |13  |190  |
|Recuperado                           |14  |216  |
|Recuperado                           |15  |195  |
+-------------------------------------+----+-----+
only showing top 20 rows
```

Figura 37. Fallecidos y recuperados por edad.

12.2.7 Caso 4. Fallecidos y recuperados por género:

```
#fallecidos y recuperados por genero
estadoGenero = data.groupby("ESTADO","SEXO").count().orderBy("SEXO")
estadoGenero.show(truncate=False)
```

```
#fallecidos y recuperados por genero
estadoGenero = data.groupby("ESTADO","SEXO").count().orderBy("SEXO")
```

```
estadoGenero.show(truncate=False)
```

```
+-------------------------------------+----+-----+
|ESTADO                               |SEXO|count|
+-------------------------------------+----+-----+
|Recuperado                           |F   |14197|
|Fallecido                            |F   |334  |
|Fallecido (No aplica No causa Directa)|F   |115  |
|Fallecido (No aplica No causa Directa)|M   |131  |
|Fallecido                            |M   |625  |
|Recuperado                           |M   |14598|
+-------------------------------------+----+-----+
```

Figura 38. Fallecidos y recuperados por genero

12.3 Visualización en gráficos de los casos analizados:

Para realizar esta parte de visualización de los datos utilizaremos una librería de pandas mediante la sentencia, y en esta misma forma convertiremos los *dataframes* de *spark* a *dataframes* de pandas, únicamente para lectura inicial.

```
import pandas as pd
fallecidosPd = fallecidos.toPandas()
tipoContagioPd = tipoContagio.toPandas()
estadoPd = estado.toPandas()
estadoGeneroPd = estadoGenero.toPandas()
```

Figura 39.Convirtiendo DataFrames spark a Dataframes de Pandas .

12.3.1 Caso 1. Fallecidos y recuperados por localidad por COVID 19:

```
fallecidosPd.groupby(['ESTADO',
'LOCALIDAD_ASIS'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Estado contagio por
localidad', kind = 'bar')
```

```
fallecidosPd.groupby(['ESTADO', 'LOCALIDAD_ASIS'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Estado contagio por localidad', kind = 'bar')
```

```
<AxesSubplot:title={'center':'Estado contagio por localidad'}, xlabel='LOCALIDAD_ASIS'>
```

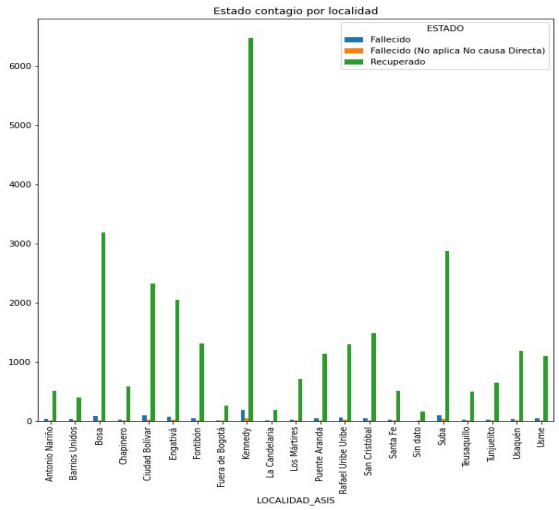


Figura 40.Grafica fallecidos y recuperados por Covid19.

12.3.2 Caso 2. Fuente tipo de contagio por edad:

```
tipoContagioPd.groupby(['FUENTE_O_TIPO_DE_CONTAGIO', 'EDAD'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Fuete de contagio', kind = 'line')
```

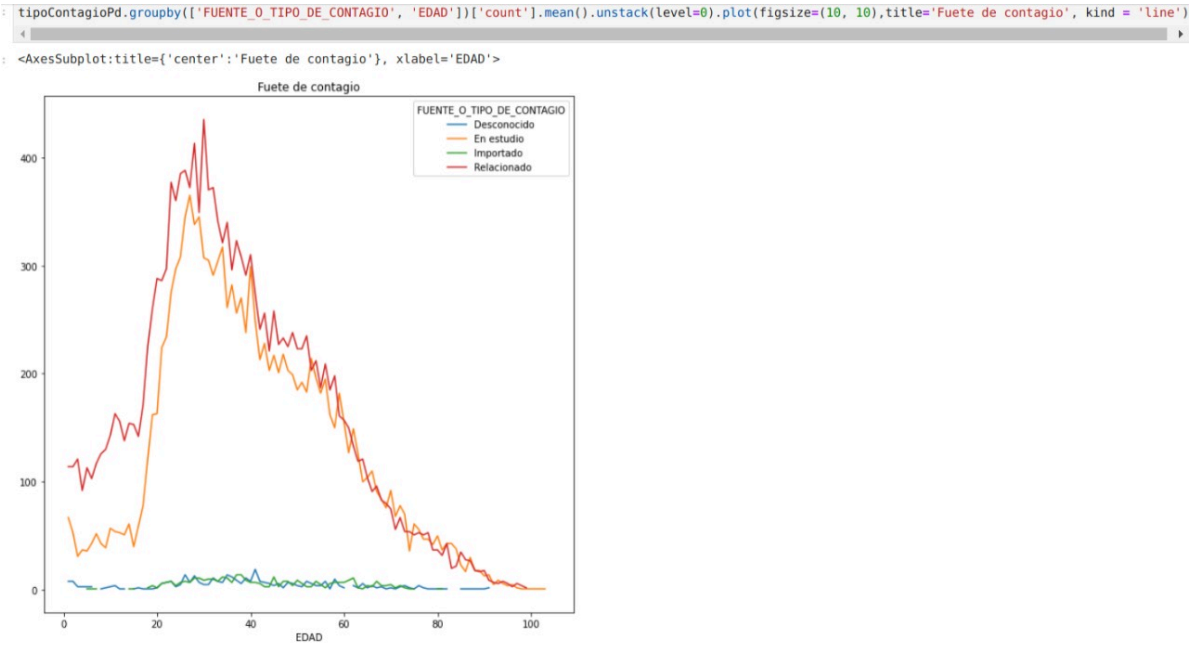


Figura 41. Grafica fuente tipo de contagio por edad.

12.3.3 Caso 3. Fallecidos y recuperados por edad:

```
estadoPd.groupby(['ESTADO', 'EDAD'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Estado de contagio por edad', kind = 'area')
```

```
[75]: estadoPd.groupby(['ESTADO', 'EDAD'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Estado de contagio por edad', kind = 'area')
[75]: <AxesSubplot:title={'center':'Estado de contagio por edad'}, xlabel='EDAD'>
```

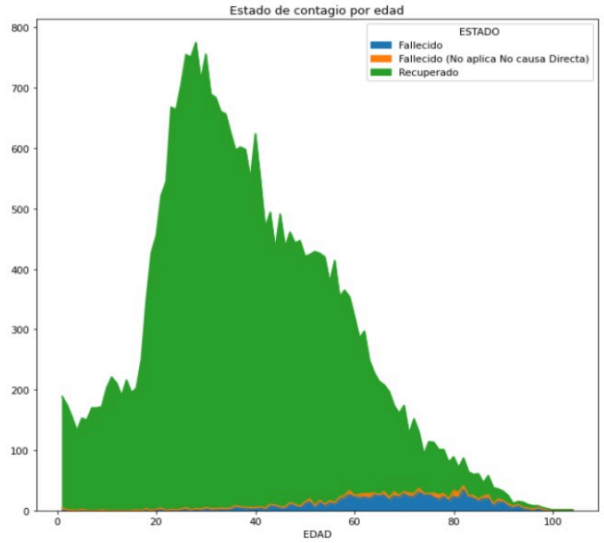


Figura 42. Grafica estado contagio por edad

12.3.4 Caso 4. Fallecidos y recuperados por género:

```
estadoGeneroPd.groupby(['ESTADO', 'SEXO'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Estado de contagio por sexo', kind = 'barh')
```

```
: estadoGeneroPd.groupby(['ESTADO', 'SEXO'])['count'].mean().unstack(level=0).plot(figsize=(10, 10),title='Estado de contagio por sexo', kind = 'barh')
: <AxesSubplot:title={'center':'Estado de contagio por sexo'}, ylabel='SEXO'>
```

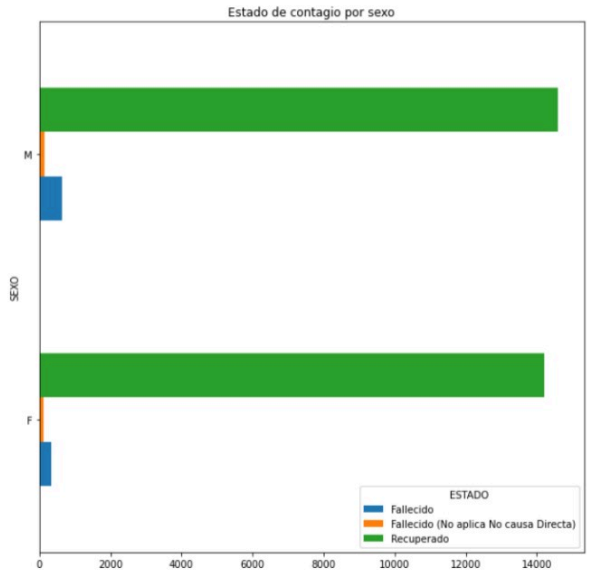


Figura 43.Grafica fallecidos y recuperados por género.

12.4 Métricas del desarrollo:

Se realizaron diferentes acciones en el entorno experimental, acciones como, *count, gruopBy, joins, show*, funciones de ventana con 30.000 y 1.000.000 de registros como se muestra en las figuras 44,45,46, realmente el entorno se comporta muy bien y es apto para que sea utilizado como ambiente de experimentación, se obtienen tiempo de respuestas óptimos, y al ser una plataforma experimental no requiere compararla con otros proyectos dado el alcance que se le está brindando.

Figura 44.Métricas de desarrollo

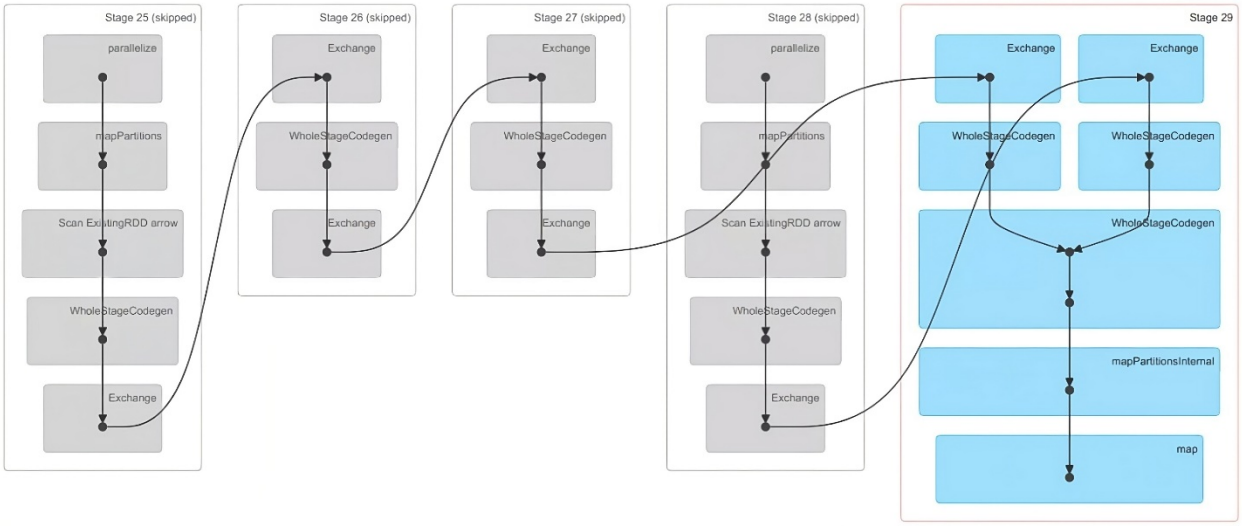


Figura 45.Visualización DAG

Figura 46.Tiempos de finalización

12.5 Ejecución devops en el ciclo de entrega de software:

Para finalizar el proceso es necesario realizar el ciclo de versionado de código y tan pronto realicemos el *push* al repositorio, ya con las configuraciones realizadas anteriormente, *gitflow* iniciará el llamado a *sonarcloud*, en el cual se realizará las validaciones de cobertura de código. Por lo que realizaremos el proceso paso a paso, recordando que la ideas es que se tenga un acercamiento experimental con flujos de trabajo.

12.5.1 Renombrado de archivo:

Cambiaremos el nombre del notebook en el cual realizamos nuestra prueba de Untitled. ipynb a Caso_de_uso.ipynb.Como lo muestra la figura 47.

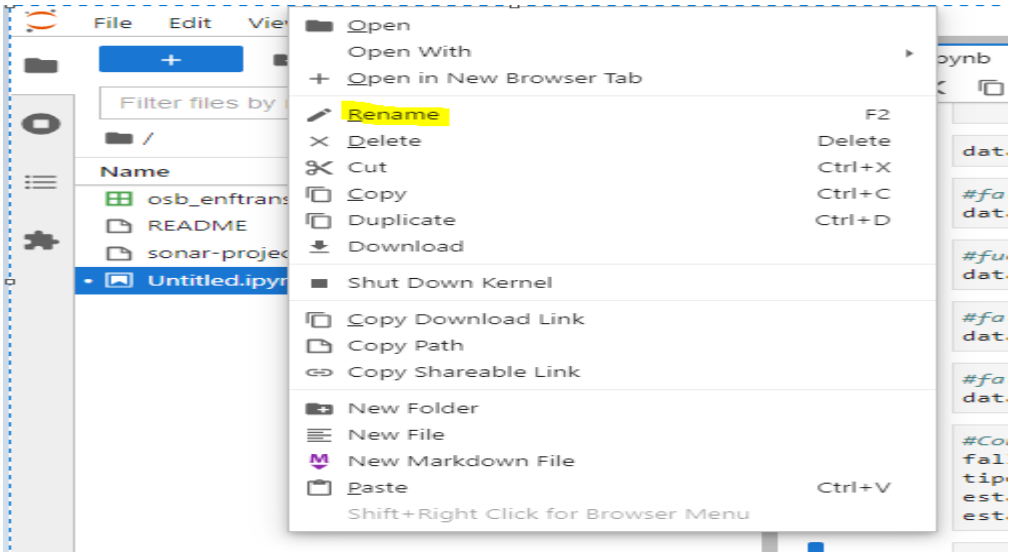


Figura 47.Renombrado de Notebook

12.5.2 Ejecución comando git mediante terminal:

Abriremos una terminal la cual nos permitirá interactuar con el ambiente en monda consola, tal cual como nos muestra la figura 48.

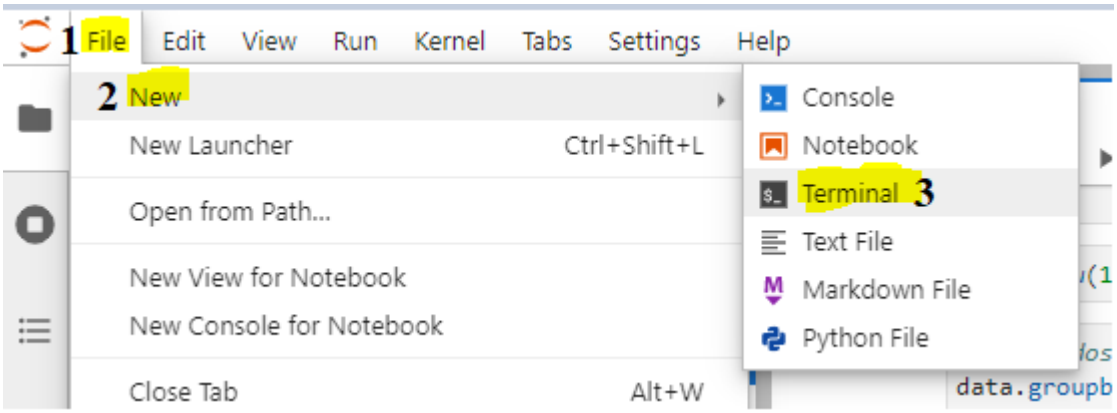


Figura 48. Abrir terminal para ejecución comandos git.

Una vez realizado este proceso se abrirá la terminal en la cual, realizaremos algunos comandos que nos permitirán verificar los archivos en este caso el notebook creado, dentro de nuestra carpeta llamada "Versionado" que es donde se inicia el ambiente y previamente se configuro el proyecto para que mediante los comandos git, podamos enviar al repositorio el desarrollo realizado y empiece el flujo de git.

Si ejecutamos el comando **"git status"** veremos las modificaciones que hay en git en color rojo como indica la figura 49. Importante agregar los archivos sonar-project.properties, y .github/

el cual contiene una configuración necesaria para la validación de condigo en sonar, el notebook que desarrollamos y si se requiere la muestra, en nuestro caso agregamos esos 3 archivos.

```
ubuntu@ip-172-31-13-42:~/Versionado$ ls
Caso_de_uso.ipynb  README  osb_enftransm-covid-19_18082023.csv  sonar-project.properties
ubuntu@ip-172-31-13-42:~/Versionado$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .github/
        .ipynb_checkpoints/
        Caso_de_uso.ipynb
        osb_enftransm-covid-19_18082023.csv
        sonar-project.properties

nothing added to commit but untracked files present (use "git add" to track)
ubuntu@ip-172-31-13-42:~/Versionado$
```

Figura 49. Git status

Para agregar cada archivo realizaremos las siguientes sentencias:

git add Caso_de_uso.ipynb

git add osb_enftransm-covid-19_18082023.csv

git add sonar-project.properties

git add .github/

Si ejecutamos el comando git status nuevamente deberán aparecer los archivos agregados en verde igual a la figura 50.

```
ubuntu@ip-172-31-13-42:~/Versionado$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   Caso_de_uso.ipynb
        new file:   osb_enftransm-covid-19_18082023.csv
        new file:   sonar-project.properties
        new file:   .github/workflows/SonarQube.yml
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .ipynb_checkpoints/

ubuntu@ip-172-31-13-42:~/Versionado$
```

Figura 50.Archivos agregados para versionado git

Ahora guardaremos nuestros cambios en nuestro repositorio local para luego enviarlos al mediante un *push* a él *hub* remoto. Con el comando **git commit –m"caso de uso"**. Ilustración figura 51.

```
ubuntu@ip-172-31-13-42:~/Versionado$ git commit -m"coso de uso"
[main f7790b8] coso de uso
 3 files changed, 30175 insertions(+)
 create mode 100644 Caso_de_uso.ipynb
 create mode 100644 osb_enftransm-covid-19_18082023.csv
 create mode 100644 sonar-project.properties
ubuntu@ip-172-31-13-42:~/Versionado$
```

Figura 51. Commit archivos repositorio local

Finalmente enviaremos mediante el comando **git push origin main**, los cambios al repositorio en la nube donde se realizará el versionado y algunas validaciones de código. En al algún caso pedirá autenticación como en otros no, recordar que si le pide que no es lo ideal en la contraseña colocar el access token generado anteriormente. Tener en cuenta que el flujo de integración corre únicamente en la rama **main**.

```
ubuntu@ip-172-31-13-42:~/Versionado$ git push origin main
Username for 'https://github.com': charswat
Password for 'https://charswat@github.com':
Counting objects: 5, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 246.69 KiB | 3.79 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/charswat/prueba.git
   beeadea..f7790b8  main -> main
```

Figura 52. Envío de cambios al repositorio en la nube.

Procedemos a validar los cambios en el repositorio, como se puede observar en la figura 90, el código mediante el *push* sube al repositorio y nos debemos centrar en las 2 pestañas resaltadas en amarillo, código, hace referencia a la imagen que observamos.

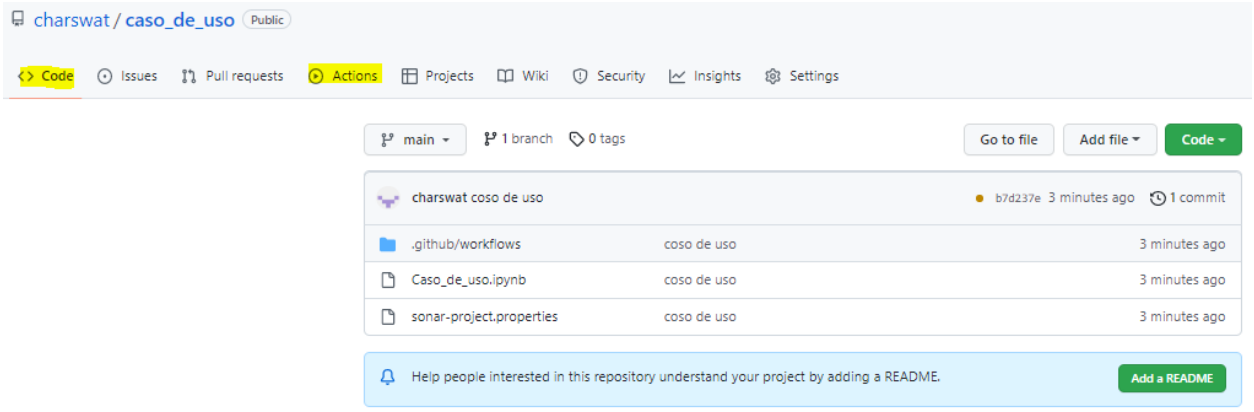


Figura 53.Código versionado en el repositorio.

Actions, es donde se compilará el flujo de validación de código mediante sonar como se observa en la figura 54. El cual al dar clic en cada uno de ellos tendremos acceso al log de las validaciones.

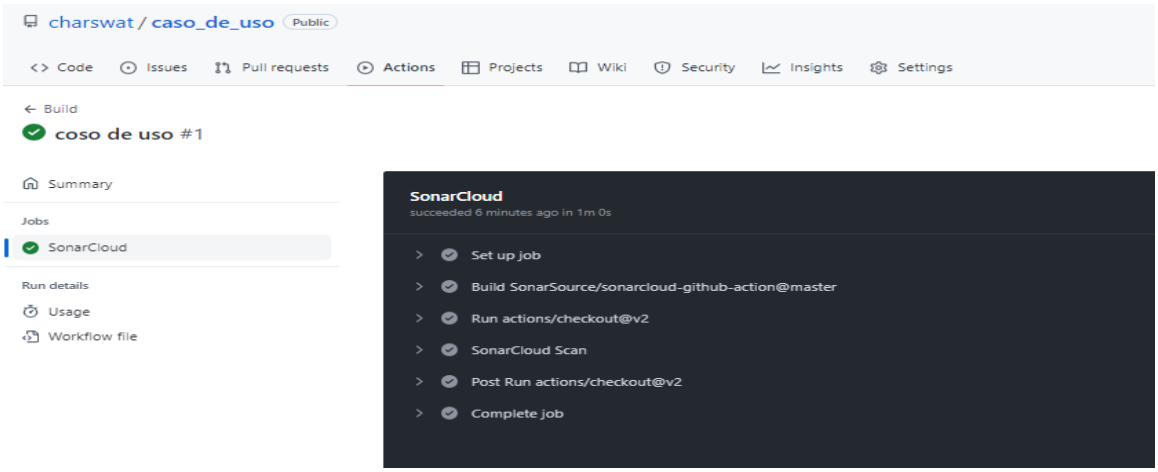


Figura 54.Validación código SanarCloud mediante GitAction

Para verificar el análisis de código de manera gráfica, abrimos sonar cloud y en el proyecto que se crea por rama del repositorio que deseamos aplicarle validaciones, se obtendrá una visual del análisis realizado.

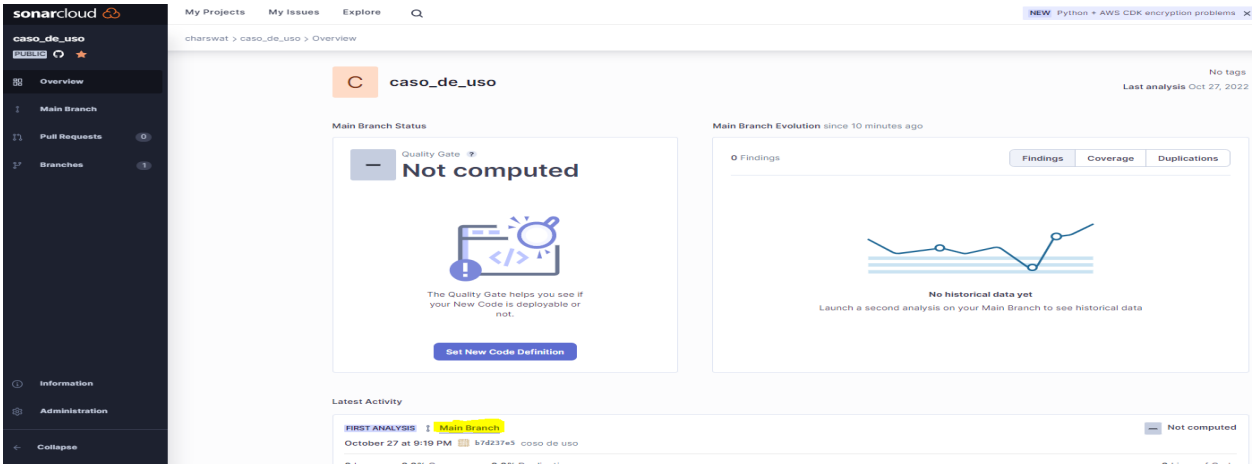


Figura 55. Gráficas de resultados sonarCloud

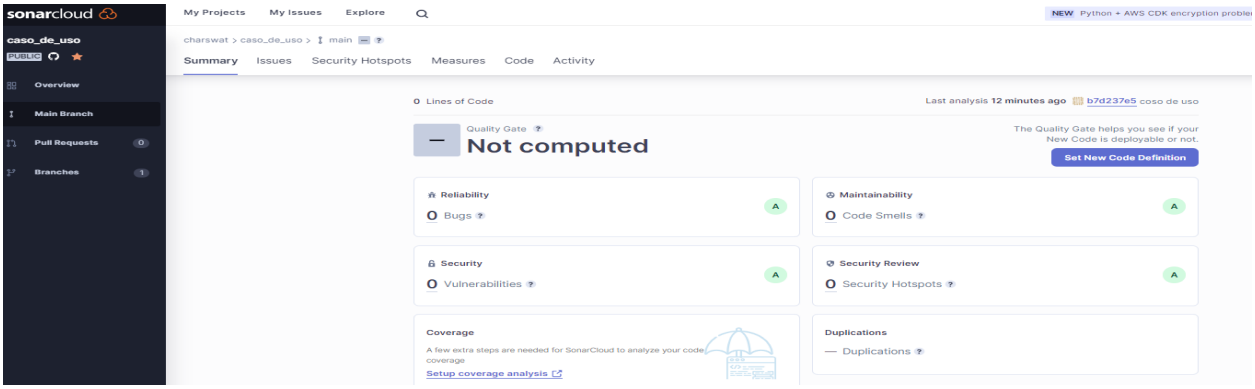


Figura 56.Resultados pruebas realizadas

La opción de incluir herramientas *devops en* el entorno, favorece a los *cloud y data* Engineering, permitiéndoles interactuar con estas herramientas ofrecidas en la nube como lo son sonarCloud y git y su *marketplace* de git action para automatizaciones de diversos proceso en flujos de trabajo en git, desarrollando habilidades de versionado de código, familiarizándose con términos como calidad de código, cobertura del mismo, *commits*, *pull request*, entre otros en la búsqueda de ofrecerles experimentar un símil de lo que se encontraran.

Luego de esto para terminar el *clúster* mediante los comandos de *terraform Destroy*, terminara todas las instancias y recursos utilizados en *Aws*.

12.6 Objetivos cumplidos:

Al final de la implementación se cumple con el objetivo de disponer un entrono experimental de procesamiento de datos distribuidos integrando *devops* en el ciclo de entrega de software, cumpliendo con cada uno de sus objetivos específicos de analizar arquitecturas de referencia en temas de procesamiento de datos y *devops*, diseñando y construyendo una arquitectura de

procesamiento de datos distribuidos e implementando el caso de uso, para corroborar su funcionamiento. Con el fin de que los futuros ingenieros de datos y los ingenieros en la nube logren enfocar sus esfuerzos de aprendizaje en la práctica de manipulación de datos y el uso de tecnologías en la nube.

12.7 Ventajas:

El ambiente está construido con tecnologías demandadas y utilizadas en el mercado, logra desplegar ágilmente aproximadamente 10 min, y desmontar en menos de 2 minutos mediante infraestructura como código(*IaC*) en la nube, minimiza el esfuerzo y errores en configuraciones manuales, siempre se tendrá disponible el mismo entorno con un clúster de procesamiento ideal para la práctica en manipulación, análisis y visualización de datos. Este último punto importante para lograr tener una experiencia visual de lo desarrollado mediante una representación de los datos y manipulación del entorno mismo. Funcional desde cualquier sistema operativo que sea compatible con *terraform*, dado a que su infraestructura de despliega en la *Aws*.

12.8 Limitaciones:

El entorno cuenta con un lenguaje definido de análisis y manipulación de los datos como lo es Python, el script solo se probó en 2 ambientes de especificaciones similares, no se realizaron pruebas del resultado de modificar el clúster, se garantizó el funcionamiento correcto del entorno hasta con 1.000.000 de registros, aunque se realizaron pruebas con 3.000.000 de registros y funciono correctamente, solo que no se plasmaron en el documento. Y finalmente no es una infraestructura elástica u escalable.

13. CONCLUSIONES

En este documento se muestra la implementación y puesta en marcha de una arquitectura y plataforma orientada la experimentación y practica en manipulación y procesamiento de datos distribuidos integrado herramientas *devops*, empleando tecnologías demandadas en el mercado, un entorno que permite a los usuarios ser competitivos ante la interacción , manipulación y práctica con un lenguaje de programación como *python*, clúster de procesamiento de datos (*Spark y hadoop*), herramientas *Devops* el ciclo de entrega de software, como es *SonarCloud* y *Github* permitiendo experimentar a los estudiante un símil de lo que se encontraran, cuando realicen desarrollos en espacios colaborativos laboralmente, utilizando tecnologías como infraestructura como código(*terraform*) y la nube mediante *Aws*, para interactuar y consumir una variedad de servicios que la *Cloud* ofrece; y que en el sector tecnológico son habilidades que requieren las

empresas en los perfiles de alta demanda. Todos los aspectos mencionados, convierten a esta plataforma en una alternativa para el aprendizaje e interacción con tecnologías líderes en el sector tecnológico.

Adicional el poder desplegar la infraestructura como código del entorno de manera ágil, realmente genera valor, y evita el hecho de configurar manualmente todos los recursos necesarios que se crean en *la nube,* se realiza la configuración del clúster automáticamente en cuestión de minutos y así mismo se destruye para no generar cobros adicionales en los servicios requeridos en *Aws*.

Sin duda la maestría nos brindó la base del conocimiento para poder profundizar y lograr crear esta solución, la cual ayudo a profundizar aún más en herramientas, arquitecturas de procesamiento de datos, patrones de diseño y desarrollo de software mediante infraestructura como código y otras tecnologías que continúan creciendo en el mercado, en donde cada problema presentado en la configuración del clúster, la creación de recursos en la nube y así su optimización, la parametrización del *notebook* y el flujo *devops*, él envió de archivos de configuración a la nube, y otros muchos inconvenientes presentados mediante lenguaje de configuración *HashiCorp Configuration Language*, nos permitió obtener un sin número de habilidades a la hora de enfrentarse con soluciones *data processing, cloud. Devops* y modelos de despliegue como código.

14. TRABAJOS FUTUROS

Para lograr que el entorno experimental sea funcional en otros ambientes diferente al desarrollado sería necesario realizar más pruebas para garantizar la generalidad del mismo.

Por otro lado, lograría ser interesante realizar la infraestructura bajo una arquitectura de contenedores y virtualización. En la parte de visualización mediante el *notebook*, resultaría útil agregar el *kernel* de *scala* para que se tenga también la facilidad de desarrollar bajo este lenguaje de programación, teniendo en cuanta que junto a *Python* son los lenguajes más demandados en lo que a manejo de datos se trata; e incluso escala esta embebido en algunas tecnologías lideres para el procesamiento de datos.

15. ANEXOS

Metodología, presupuesto y cronograma.

Retroalimentación público objetivo.

Manual entorno experimental Procesamiento de datos distribuidos.

Desarrollo en repositorio git.

https://github.com/charswat/Entorno-experimental-de-procesamiento-de-datoscon-devops-bajo-un-despliegue-de-IaC

16. BIBLIOGRAFÍA

¿Qué es la infraestructura como código - Infrastructure as Code? (n.d.). Retrieved March 14, 2023, from https://www.redhat.com/es/topics/automation/what-is-infrastructure-as-code-iac

Ali, M., Khan, S. U., & Vasilakos, A. V. (2015). Security in cloud computing: Opportunities and challenges. *Information Sciences*, *305*, 357–383. https://doi.org/10.1016/j.ins.2015.01.025

Ashraf, I. (2014). An Overview of Service Models of Cloud Computing. *International Journal of Multidisciplinary and Current Research*, *2*(August 2014), 779–783. http://ijmcr.com/wp-content/uploads/2014/08/Paper18779-783.pdf

Bello, E. (2022). ¿Qué es Data Engineering? Funciones, requisitos y salario. *Thinking for Innovation*. https://www.iebschool.com/blog/data-engineering-big-data/

Bhimani, J., Yang, Z., Leeser, M., & Mi, N. (2017). Accelerating big data applications using lightweight virtualization framework on enterprise cloud. *2017 IEEE High Performance Extreme Computing Conference, HPEC 2017*. https://doi.org/10.1109/HPEC.2017.8091086

Blazquez, D., & Domenech, J. (2018). Big Data sources and methods for social and economic analyses. *Technological Forecasting and Social Change*, *130*(March 2017), 99–113. https://doi.org/10.1016/j.techfore.2017.07.027

Data Warehousing in the Age of Big Data - Krish Krishnan - Google Libros. (n.d.). Retrieved April 12, 2022, from https://books.google.com.co/books?hl=es&lr=&id=8ngws8f_lNsC&oi=fnd&pg=PP1&dq=Data+Warehousing+in+the+Age+of+Big+Data.+In+Data+Warehousing+in+the+Age+of+Big+Data.+https://doi.org/10.1016/C2012-0-02737-

8&ots=gVKaeYc9iv&sig=LKkNPSvJJwmD9aM4rdzhNVPdkQM&redir_esc=y#v=onepage&q&f=false

Gartner. (2021). *Gartner Reprint*. https://www.gartner.com/doc/reprints?id=1-254T1IQX&ct=210202&st=sb

Gartner. (2022). *Gartner Reprint*. https://www.gartner.com/doc/reprints?id=1-292LEME3&ct=220209&st=sb

Golfarelli, M., & Rizzi, S. (2020). A model-driven approach to automate data visualization in big data analytics. *Information Visualization*, *19*(1), 24–47. https://doi.org/10.1177/1473871619858933

Gonçalves, A., Portela, F., Santos, M. F., & Rua, F. (2017). Towards of a Real-time Big Data Architecture to Intensive Care. *Procedia Computer Science*, *113*, 585–590. https://doi.org/10.1016/j.procs.2017.08.294

Gupta, M., Chowdary, M. N., Bussa, S., & Chowdary, C. K. (2021). Deploying Hadoop Architecture Using Ansible and Terraform. *2021 5th International Conference on Information Systems and Computer Networks, ISCON 2021*, 1–6. https://doi.org/10.1109/ISCON52037.2021.9702299

Howard, C. (2020). *Top Priorities for IT: LeadershipVision for 2021, Data and Analytics Leaders*. gartner.com

L., Camargo, V., Camargo-Ortega, J. J., & Joyanes-Aguilar;, J. F. . (2015). *Vista de Arquitectura vertida*. *1*, 7–18. https://doi.org/https://doi.org/10.14483/udistrital.jour.RC.2015.21.a1

Naik, N. (2017). Docker container-based big data processing system in multiple clouds for everyone. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*. https://doi.org/10.1109/SysEng.2017.8088294

Osorio, G. A., Del Real, C. S., Valdez, C. A. F., Miranda, M. C., & Garay, A. H. (2006). Effect of inclusion of cactus pear cladodes in diets for growing-finishing lambs in central Mexico. *Acta Horticulturae*, *728*, 269–274.

Rajesh, S., Swapna, S., & Reddy, P. S. (2012). Data as a Service (Daas) in Cloud Computing [Data-As-A-Service in the Age of Data] Data as a Service Daas in Cloud Computing Data-As-A-Service in the Age of Data Data as a Service (Daas) in Cloud Computing [Data-As-A-Service in the Age of Data]. *Global Journal of Computer Science and Technology Cloud*, *January*.

Smith, D., Villaba, D., Irvine, M., Stanke, D., & Harvey, N. (2021). *Accelerate State of DevOps 2021*. 45. https://cloud.google.com/blog/products/devops-sre/announcing-dora-2021-accelerate-state-of-devops-report

Sousa, T., Ferreira, H. S., & Correia, F. F. (2021). A Survey on the Adoption of Patterns for Engineering Software for the Cloud. *IEEE Transactions on Software Engineering*, *5589*(c), 1–13. https://doi.org/10.1109/TSE.2021.3052177

Technology, I. C. (2015). *Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery*. *Intech*, 78–82.

What is a Cloud Engineer and How Do You Become One? (n.d.). Retrieved March 14, 2023, from https://www.techtarget.com/searchcloudcomputing/definition/cloud-engineer

Zhelev, S., & Rozeva, A. (2017). Big data processing in the cloud - Challenges and platforms. *AIP Conference Proceedings*, *1910*(December 2017). https://doi.org/10.1063/1.5014007

宗成庆. (2021). *State of Software development*. 48.