ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE SERVICIOS DE NETWORKING MEDIANTE DEVOPS

IMPLEMENTACIÓN DE KAFKA MEDIANTE DEVOPS

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN REDES Y TELECOMUNICACIONES

GEOMAIRA ELIZABETH PASTO CHAMBA
geomaira.pasto@epn.edu.ec

DIRECTOR: FERNANDO VINICIO BECERRA CAMACHO
fernando.becerrac@epn.edu.ec

DMQ, febrero 2024

CERTIFICACIONES

Yo, GEOMAIRA ELIZABETH PASTO CHAMBA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

GEOMAIRA PASTO

geomaira.pasto@epn.edu.ec

geomairapasto@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por GEOMAIRA ELIZABETH PASTO CHAMBA, bajo mi supervisión.

FERNANDO BECERRA

DIRECTOR

fernando.becerrac@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

GEOMAIRA PASTO

DEDICATORIA

Esta tesis está dedicada con profundo agradecimiento y amor.

A Dios, por brindarme la fortaleza para superar cada obstáculo, la sabiduría para crecer y la salud para continuar adelante.

A mis padres, José y Rosa, cuyo incansable esfuerzo, amor incondicional, sacrificio y apoyo constante han sido el motor que me impulsó a alcanzar esta meta. Gracias por la paciencia y guía en este viaje los quiero.

A mi querida hermana, María de los Ángeles, por su inquebrantable apoyo y por ser mi compañía de noches de desvelo a lo largo de este camino.

A mi tía Clara y mi primo Jefferson, por su presencia constante desde mi infancia hasta este día.

A mis adoradas mascotas, Osito, Benji, Sultán y Jack (que en paz descanse), quienes fueron mis leales compañeros, llenando mis días con alegría y sacando sonrisas en momentos de tristeza.

Por último, dedico esta tesis a mis amigos, por su apoyo incondicional, por compartir conmigo las experiencias vividas y por brindarme su alegría en los momentos más difíciles. Su amistad ha sido un regalo invaluable en este viaje.

Geomaira Elizabeth

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a la Escuela Politécnica Nacional, en particular a la Escuela de Formación de Tecnólogos, por brindarme la invaluable oportunidad de transitar por sus aulas y alcanzar este significativo logro académico.

A mis queridos padres, gracias por todo el sacrificio y amor. Este logro es para ustedes.

A mi tutor, Fernando Becerra, por su inestimable paciencia y guía durante todo el proceso para el desarrollo de esta tesis.

Al Ing. Rodrigo Luna, especialista TIC de la DGIP, cuya dedicación y sabiduría han sido una inspiración constante. Sus enseñanzas han enriquecido enormemente mi experiencia académica.

A mis queridos amigos, Mayerli, Adriana, Alex y Benedict, les agradezco profundamente por su inquebrantable apoyo, comprensión y palabras de aliento que me han sostenido en los momentos más desafiantes de este viaje académico.

Finalmente, extiendo mi gratitud a Eternit Ecuatoriana S.A. por brindarme la oportunidad de crecimiento profesional. Especialmente, agradezco al MSc. Cristian Herrera Terán por ser un excelente mentor, por sus valiosos consejos, enseñanzas, por abrirme las puertas del mundo laboral y por su constante respaldo y orientación. También quiero reconocer la bonita amistad de Lizeth C., Anita E. y Gabriela T., quienes hicieron mi estadía en Eternit más enriquecedora y gratificante.

A todas estas personas e instituciones, mi más profundo agradecimiento por su contribución a este logro que marca un hito en mi trayectoria académica y profesional.

Geomaira Elizabeth

ÍNDICE DE CONTENIDOS

RESUMEN

En este trabajo de integración curricular, se lleva a cabo la automatización de los servicios de *networking* utilizando *Ansible*. Se emplea un *playbook* para ejecutar tareas en el dispositivo final, que es el cliente donde se implementará el servicio de *Kafka*. El proyecto está organizado en cuatro secciones que abarcan detalles sobre la implementación de servicios de *networking* a través de la metodología *DevOps*.

En la primera sección del documento, se presentan los objetivos que abarcan el proyecto a desarrollar. Además, se incluyen los conceptos teóricos para comprender la implementación que se llevará a cabo sobre *DevOps, Ansible, Playbook y Kafka*.

En la segunda sección, se expone la metodología diseñada para garantizar el cumplimiento de los objetivos establecidos en la primera sección. Aquí se describen detalladamente los procedimientos que se han seguido para llevar a cabo la implementación.

En la tercera sección, se presentan los resultados obtenidos acorde a los objetivos que previamente han sido definidos. Se detallan los requerimientos necesarios para elaborar el *playbook* en *Ansible*, destinado a la implementación de servicios de *networking* como *Kafka* y *Zookeeper*. En este proceso se va a considerar el sistema operativo y las dependencias que requiere para la implementación.

Por último, se encuentra la sección final del documento en dónde se encuentran las pruebas del correcto funcionamiento del *playbook* desarrollado.

En el caso de evidenciar fallas de funcionamiento, se dará la solución que solventará el problema.

PALABRAS CLAVE: *DevOps, Ansible, Playbook, Kafka*

ABSTRACT

In this curriculum integration work, the automation of networking services is carried out using Ansible. A playbook is used to execute tasks on the end device, which is the client where the Kafka service will be deployed. The project is organized in four sections covering details on the implementation of networking services through DevOps methodology.

In the first section of the document, the objectives covering the project to be developed are presented. In addition, the theoretical concepts to understand the implementation to be carried out on DevOps, Ansible, Playbook and Kafka are included.

In the second section, the methodology designed to guarantee the fulfillment of the objectives established in the first section is exposed. Here, the procedures followed to carry out the implementation are described in detail.

In the third section, the results obtained according to the previously defined objectives are presented. The necessary requirements to elaborate the playbook in Ansible, destined to the implementation of networking services such as Kafka and Zookeeper, are detailed. In this process, the operating system and the dependencies required for the implementation will be considered.

Finally, there is the final section of the document where the tests of the correct operation of the developed playbook are found.

In the case of evidencing malfunctions, the solution that will solve the problem will be given.

KEYWORDS *DevOps, Ansible, Playbook, Kafka.*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El presente proyecto tiene como objetivo demostrar el funcionamiento de *Ansible*, mediante las tareas de automatización desarrolladas para la instalación del servicio de *Kafka*. En particular, se emplea *Ansible* para automatizar el despliegue de los servicios de *networking*.

El despliegue de este componente implica aprovechar las herramientas que se encuentran disponibles en *Ansible*, como el inventario y el *playbook*. Estas herramientas trabajarán en conjunto para permitir la configuración remota desde el servidor hacia el cliente, lo que facilita la implementación del servicio de *Kafka* en la máquina designada como cliente.

1.1 Objetivo general

Implementación de servicios de *networking* mediante *DevOps*.

1.2 Objetivos específicos

- Analizar la herramienta de DevOps, herramientas de automatización y Kafka.
- Diseñar la solución para cada servicio de *networking* mediante herramientas de *DevOps*.
- Implementar las soluciones mediante herramientas de *DevOps* para el despliegue de los servicios de *networking*. Se implementará el *playbook* encargado del despliegue de *Kafka*
- Verificar el funcionamiento de cada servicio de *networking* implementado mediante *DevOps*.

1.3 Alcance

Por medio del presente proyecto se busca implementar servicios de *networking* mediante *DevOps*, facilitando la automatización remota con ayuda de *Ansible*. Esto permitirá al usuario realizar configuraciones de forma remota, sin la necesidad de estar físicamente presente en el equipo. Como resultado, el usuario podrá llevar a cabo las siguientes acciones:

- Realizar actualizaciones del sistema mediante CLI.
- Creación de archivos mediante CLI.
- Descargas mediante enlaces por CLI.

- Activación de servicios.

1.4 Marco Teórico

DevOps

DevOps es una metodología que fusiona modelo de desarrollo de *software* (*Dev*) y operaciones de *TI* (*Ops*), donde se destacan los procesos continuos de integración. Este enfoque describe un proceso de desarrollo de *software* y un cambio de la cultura organizacional, el cual acelera la entrega de *software* o servicios de mayor calidad al automatizar e integrar los equipos de desarrollo y operaciones de *Information Technology (TI).* El modelo *DevOps* se encuentra estructurado en fases que incluyen planificación, construcción, pruebas de implementación, registros, configuración, monitoreo y recopilación durante las diferentes fases de desarrollo. Estas fases fomentan la automatización de procesos [1].

Ansible

Ansible es una de las herramientas de *software* ampliamente utilizada para automatizar procesos, basada en el lenguaje de programación *Python* y de código abierto, lo que la hace altamente accesible [2]. *Ansible* emplea por defecto el protocolo *Secure Socket Shell (SSH)* para conectase a servidores y ejecutar las tareas. En *Ansible*, se utilizan módulos, también conocidos como *playbooks*, que están escritos en formato *Yet Another Markup Language (YAML).* Estos *playbooks* requieren de directrices para que pueda ejecutar cada tarea, lo que permite la creación de infraestructuras como código a través de *scripts* simples. Los módulos se definen de acuerdo con la necesidad de cada nodo a ser administrado. En el nodo controlador se encuentran el inventario y el *playbook*. El inventario contiene la lista de *hosts* donde se ejecutarán los módulos, mientras que el *playbook* contiene a las tareas a realizar. Además, *Ansible* se organiza en dos categorías de computadoras: el nodo controlador y nodos administrados. La Figura 1.1 ilustra el funcionamiento de *Ansible* [2].

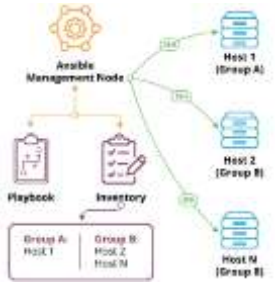

Figura 1.1 Estructura Ansible

Con *Ansible*, se facilita la automatización en varios aspectos como, la gestión de configuración, aprovisionamiento de infraestructura y nube, implementación de aplicaciones, automatizar tareas diariamente, instalación de *software*, optimizar la seguridad, administración de sistemas y muchos procesos de *TI* [3].

Playbook

Un *playbook* corresponde a scripts o archivos en formato de texto plano que tiene la extensión *YAML*. En *Ansible*, se empieza desde tres guiones porque requiere del uso de sangría y espacios y no de tabuladores el cual permite la legibilidad y escritura del usuario. Cada *playbook* describe paso a paso los procesos que se van a ejecutar en un dispositivo determinado para automatizar tareas. Puede contener una o varias tareas que van a seguir un orden ya que al ejecutar el *playbook* se hace de manera secuencial. Una *playbook* tiene la función de poder gestionar las configuraciones en servidores remotos, aplicaciones y servicios, pero también controlar entornos en sistemas operativos, redes y sistemas de seguridad. En resumen, un *playbook* tiene como finalidad mapear grupos de *hosts* [4].

Kafka

Kafka, desarrollado por *LinkedIn* en 2011, se fundamenta bajo el concepto de modelo publicación-suscripción con el propósito de ofrecer un alto rendimiento y tolerancia a fallas manejando millón de datos por segundo. Esta herramienta, catalogada como apache *Kafka* de código abierto, se define como un sistema para manipulación, transmisión e integración de datos en tiempo real. Su función principal radica en el procesamiento de gran cantidad de flujo de datos y almacenamiento de eventos de manera eficiente. Además, destaca por admitir la transmisión de datos en mayor cantidad con bajas latencias. *Kafka* se destaca como un sistema de almacenamiento y transmisión de datos en tiempo real, donde la contante generación de datos da lugar a registros continuos. El flujo de datos requiere un sistema de *streaming* capaz de procesarlos de manera secuencial y progresiva [5]. La representación visual de *Kafka* se muestra en la Figura 1.2.

Figura 1.2 Kafka *[5]*

2 METODOLOGÍA

De acuerdo con el plan de trabajo de titulación, se realiza una división por secciones, las mismas que se detallan cómo se cumplirán los objetivos planteados.

En la sección 1 comprende a la investigación bibliográfica sobre las herramientas de automatización *DevOps* y *Kafka* para recaudar información indispensable y las características para su implementación. Con la información obtenida ayudó a una mejor asimilación sobre el funcionamiento de las herramientas de automatización, y la forma de cómo se podría automatizar los servicios de *networking*. No obstante, también fue importante utilizar recursos audiovisuales que ayudaron a la investigación.

En la sección 2 corresponde a la fase del diseño para la solución del servicio de *networking* mediante *DevOps*. Se creó una máquina virtual que actuará como servidor con un sistema operativo *Ubuntu LTS* 22.04 mediante el hipervisor *VirtualBox*. En donde se realizaron pruebas sencillas del servicio de *Kafka* y *Zookeeper* se encuentren activos y ejecutándose en la máquina virtual que fue previamente configurado. Con las pruebas efectuadas, se pudo concretar el proceso para el diseño del *playbook* para la implementación del servicio *Kafka*. A continuación, también se creó una máquina virtual adicional que cumple con el rol de cliente.

En la sección 3 denominada implementar las soluciones mediante herramientas de *DevOps* para el despliegue de los servicios de *networking* . En la máquina virtual que actúa como servidor se agregó el repositorio de la herramienta *Ansible*, posteriormente se realizó la instalación del conjunto de herramientas de programación de *Phyton*. Además, se editó el archivo de *host* y se realizó una prueba para verificar la comunicación *SSH* entre en el servidor y el cliente.

En la sección 4 que corresponde a la implementación del *playbook* para el despliegue de *Kafka*, garantizada la comunicación *SSH* entre el servidor y el cliente, se define el *playbook Ansible* el mismo que contiene los comandos para la implementación del servicio de *Kafka*.

En la sección final se verificará el funcionamiento de cada servicio de *networking* implementado mediante *DevOps*. Las pruebas de funcionamiento del servicio de *Kafka* y *Zookeper* que se encuentren activos en el cliente, así como la creación de las carpetas de *Kafka*. Además, se utilizó el comando de *ping* para verificar que la transmisión de paquetes ingrese al servidor y cliente.

3 RESULTADOS

En esta sección se presenta los procedimientos para el correcto funcionamiento de la implementación del servicio de *networking*. Se realiza el diseño de la implementación del servicio de *Kafka* y *Zookeeper* en una máquina cliente el cual se va a realizar mediante un servidor usando las herramientas de *DevOps* como es *Ansible*. En *Ansible* se hace el uso de *playbooks* los cuales van a ejecutar tareas en un *host* desde un nodo controlador. Al implementar los servicios de *networking* se muestran las pruebas del funcionamiento.

3.1 Analizar la herramienta de DevOps, herramientas de automatización y Kafka

La implementación con *DevOps* conlleva múltiples beneficios que van más allá de garantizar la eficiencia y calidad del desarrollo de *software*. También abarca la automatización de los servicios de *networking* y aplicaciones. A continuación, se detallan algunas de las ventajas más destacadas.

- La automatización con *DevOps* es una poderosa herramienta diseñada para automatizar tareas manuales o repetitivas en el desarrollo de *software*. No solo facilita la eliminación de procesos tediosos, sino que también fomenta la colaboración y la comunicación al permitir que los miembros del equipo automaticen tareas rutinarias en menos tiempo y obtener un *software* controlado [6].
- *DevOps* en la velocidad de ejecución ayuda a que los procesos tales como la integración de códigos e implementación de las aplicaciones se realicen mucho más rápido. Porque no es necesario que un humano se encuentre listo para empezar con el proceso, ya que no existe demoras en estos procesos automatizados [6].
- Para lograr un monitoreo optimizado, se evalúa tanto el rendimiento como la seguridad, centrándose en la experiencia del usuario final. Estos equipos están dedicados al monitoreo de la infraestructura de *TI*, con el objetivo de medir métricas que reflejen la experiencia del usuario, generando estadísticas de producción para asegurar un funcionamiento correcto y eficiente.

El sistema *DevOps* se apoya en un conjunto de herramientas técnicas, como *Kubernetes*, *Ansible, Docker*, entre otras, que facilitan el intercambio de información entre equipos de desarrollo y operaciones. Estas herramientas son fundamentales para el monitoreo y control de errores, permitiendo una colaboración fluida y eficaz entre los distintos equipos involucrados [7].

Es necesario revisar a detalle sobre *Ansible* y su funcionamiento, porque es un elemento clave para el despliegue del *playbook*.

Ansible

Ansible es una plataforma de código abierto ampliamente adoptada para la automatización y gestión de sistemas informáticos. Facilita la automatización de aplicaciones, configuraciones, actualizaciones de servidores y otras tareas. Con *Ansible*, es posible implementar sistemas tanto en entornos *Unix* como *Windows*. Uno de los aspectos más destacados de *Ansible* es su funcionamiento sin necesidad de agentes adicionales, lo que significa que no se requieren *software* específico en los sistemas para llevar a cabo los cambios necesarios [8]. A continuación, los beneficios clave de utilizar *Ansible*:

- Disminuye el uso de recursos para la gestión de *TI*. De manera que se pueden controlar varias máquinas desde un nodo controlador y a la misma vez.
- Es accesible la automatización, mediante los *playbooks* que están en formato *YAML* ya que es un lenguaje que es comprensible para las personas.
- Utiliza el protocolo de comunicación *SSH* para establecer las comunicaciones con los *hosts* remotos [8].

Kafka

Esta plataforma se destaca por ser de código abierto y está diseñada para gestionar tanto la transmisión como el almacenamiento de datos en tiempo real. Está optimizada para soportar altas tasas de transmisión o flujo de datos con baja latencia. Utiliza un modelo de registro particionado, lo que significa que el registro se organiza en colecciones y se divide para varios suscriptores. Esto permite una mayor escalabilidad para los suscriptores en un mismo tema [9]. *Kafka* tiene el flujo de datos de tipo asincrónico y está compuesto por seis componentes que forman al sistema, a continuación, se describe cada uno de ellos:

- Productor: es quien se encarga de generar grandes cantidades de datos y los guarda en *Kafka*.
- Consumidor: va a actuar como el usuario final que se encarga de leer los datos de *Kafka*.
- Tema: es la etiqueta en donde se va a almacenar y publicar los registros.
- *Brokers*: se denominan a los servidores *Kafka* que son los que van a manejar los datos.
- Partición: es la unidad de almacenamiento de los datos. Los mensajes se van a almacenar en un registro con una identificación *ID* que es único, al que se lo

denomina desplazamiento de partición. Los temas contienen varias particiones con el propósito de manejar mayor flujo de datos.

- *Zookeeper*: corresponde a un servicio que es centralizado que tiene la finalidad de coordinar actividades de un clúster de *Kafka*, ya que es el responsable de sostener la lista de intermediarios del grupo [9].

3.2 Diseñar la solución para cada servicio de networking mediante herramientas de DevOps

Creación de máquinas virtuales

Se crearon dos máquinas virtuales en el hipervisor *VirtualBox*, las cuales van a cumplir con la función de servidor y cliente. Para lo cual se utilizó el sistema operativo *Ubuntu LTS* 22.04 ya que ofrece una interfaz de fácil uso y administración, y al ser compatible para los servicios de *Kafka* y *Zookeeper*. Tanto para el servidor como para el cliente se establecieron direcciones *IP* fijas con la finalidad de garantizar el acceso, la identificación de estos y que estos se encuentren disponibles en una misma dirección evitando los cambios de direcciones *IP*. Para el servidor como para la máquina de cliente. A continuación, se presenta las características detalladas para el servidor en la Figura 3.1 y para la máquina cliente en la Figura 3.2. En la máquina virtual que corresponde al servidor se va a instalar la herramienta *Ansible*, además es importante mencionar que en esta máquina se realizó la instalación del servicio de *networking* como *Kafka* y *Zookeeper* previamente a su ejecución.

Figura 3.1 Características máquina virtual Servidor Ubuntu

Figura 3.2 Características máquina virtual Cliente Ubuntu

Instalación de Kafka y Zookeeper

Se inicia con los pasos previos para la instalación de *Kafka* y *Zookeeper*. Previamente, se realiza la actualización de los repositorios del sistema con el comando sudo apt-get update tal como se muestra en la Figura 3.3. Para mantener las versiones actualizadas de los paquetes en el sistema.

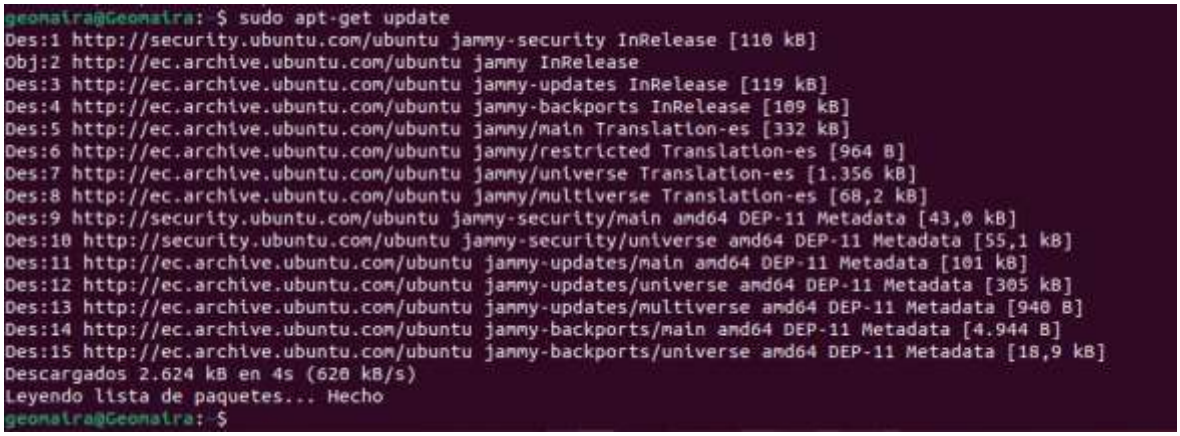

Figura 3.3 Actualización de repositorios

Una vez actualizado los repositorios se instala *OpenJDK-11* con el comando sudo apt install openjdk-11-jdk como se ve en la Figura 3.4 para el desarrollo de aplicaciones que utiliza *Kafka* y que dependen de *Java* para sus sistemas y servicios. El que proporcionará un entorno compatible para construir sistemas de transmisión en tiempo real de datos.

Figura 3.4 Instalación de OpenJDK 11

Se evidencia en la Figura 3.5 que ha terminado la instalación de *OpenJDK 11*, correctamente.

Figura 3.5 Instalación terminada de OpenJDK 11

Para verificar que se ha instalado correctamente *Java* en el sistema, se utiliza el comando java-version como se evidencia en la Figura 3.6.

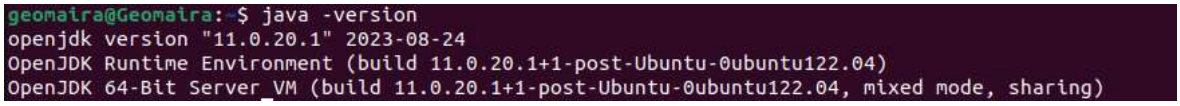

Figura 3.6 Verificación de Java instalado

Cuando ya haya instalado *Java* en el sistema, se procede a descargar *Kafka* de tipo binario a través del sitio web de *Kafka*. Para lo cual se realiza la descarga mediante el comando sudo wget https://downloads.apache.org/kafka/3.6.0/kafka_2.13-3.6.0.tgz, la función del comando es descargar archivos sin la necesidad de acceder a un navegador web, tal como se muestra en la Figura 3.7.

Figura 3.7 Descarga del binario Kafka

Se procede a verificar que se ha descargado el archivo comprimido de *Kafka* en formato tar.gz y se va a ejecutar el comando ls, el que permite enlistar a los archivos y directorios como se ve en la Figura 3.8.

Figura 3.8 Verificación de archivos y directorios

Cuando se ha verificado que el archivo comprimido se encuentra descargado en el sistema, se procede a descomprimir el archivo comprimido en el formato tar.gz y dar permisos con el comando sudo tar xzf Kafka_2.13-3.6.0.tgz, a continuación, se ejecuta el comando ls para verificar que se haya descomprimido como se evidencia en la Figura 3.9. Los permisos asignados son: "x" con el cual se va a extraer archivos del archivo que está en formato tar; "z" va a indicar que el archivo de tipo tar se va a comprimir en formato gzip y "f" se refiere a como se encuentra nombrado el archivo de tipo tar.

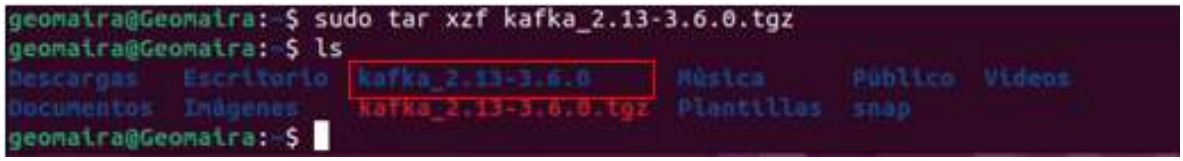

Figura 3.9 Archivo descomprimido

Cuando se ha descomprimido el archivo se va a mover a otra ubicación a un nuevo directorio nombrado como /opt/kafka ejecutando el siguiente comando sudo mv kafka_2.13-3.6.0 /opt/Kafka como la Figura 3.10, para posteriormente verificar que se ha movido al directorio especificado se coloca el comando ls.

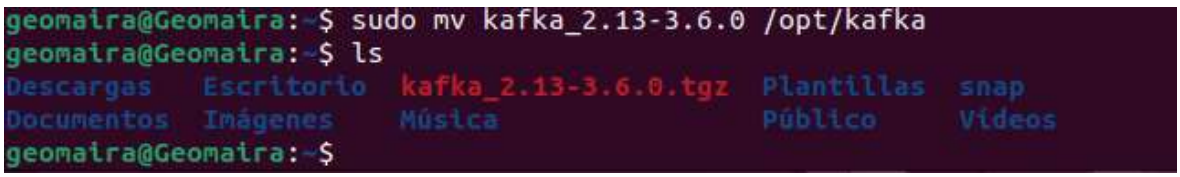

Figura 3.10 Archivo descomprimido reubicado de directorio

Ahora, se procede a la creación de archivos de unidad *systemd* para el servicio de *Zookeeper* y *Kafka*. La unidad *systemd* se denomina al conjunto de herramientas, librerías, servicios y demonios o *daemons* que se encuentran diseñados para la administración y configuración de los servicios en los sistemas operativos como *Linux*. Además, permiten crear servicios para iniciarlo, detenerlo cuando lo deseemos [10]. Para lo cual se crea el archivo de *Zookeeper* con el siguiente comando sudo nano /etc/systemd/system/zookeeper.service como se ve la Figura 3.11.

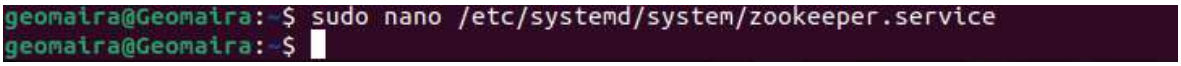

Figura 3.11 Creación de archivo systemd para Zookeeper

Al utilizar el comando nano se va a abrir el archivo para la configuración en el cual se va a escribir las siguientes líneas para posteriormente guardarlo tal como se muestra en la Figura 3.12.

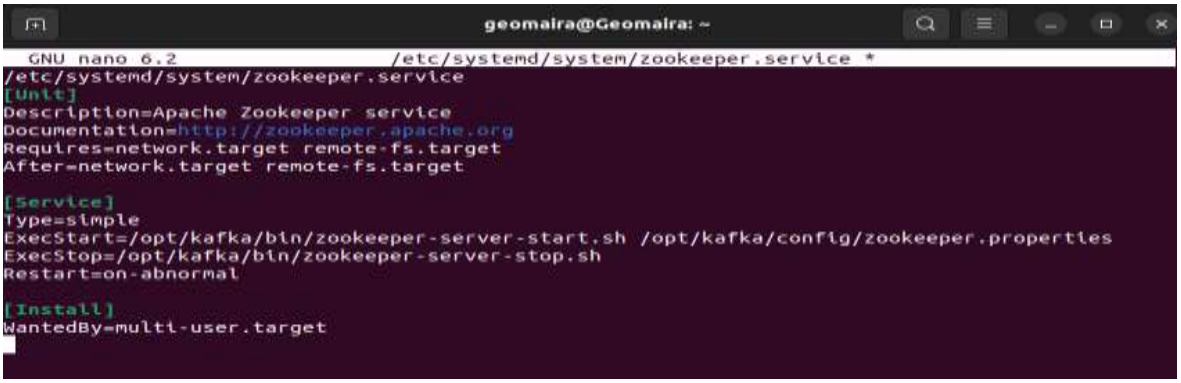

Figura 3.12 Edición del archivo systemd Zookeeper

Como siguiente paso se va a volver a cargar el demonio o *daemon* con el comando sudo systemctl daemon-reload como se evidencia en la Figura 3.13.

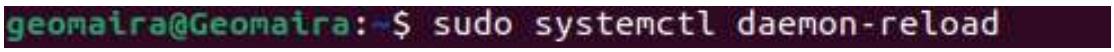

Figura 3.13 Daemon de Zookeeper recargado

Ahora, se va a crear el archivo *systemd* para *Kafka* con el siguiente comando sudo nano /etc/systemd/system/kafka.service como se ve la Figura 3.14.

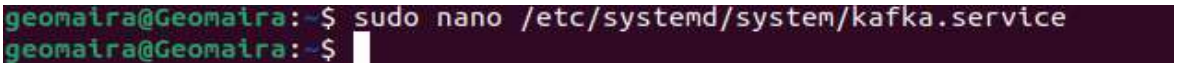

Figura 3.14 Creación de archivo systemd para Kafka

De igual manera al usar *nano* se va a editar el archivo con las siguientes líneas como en la Figura 3.15.

Figura 3.15 Edición del archivo systemd Kafka

Posteriormente se procede a cargar de nuevo al *daemon* con el siguiente comando sudo systemctl daemon-reload como se evidencia en la Figura 3.16.

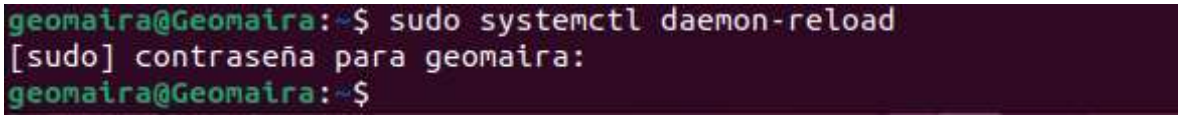

Figura 3.16 Daemon de Kafka recargado

Una vez que ya se ha recargado los dos *daemons* tanto *Kafka* como *Zookeeper* se procede a iniciar los dos servicios, empezando por *Zookeeper* ejecutando el comando sudo systemctl start zookeeper tal como se muestra en la Figura 3.17.

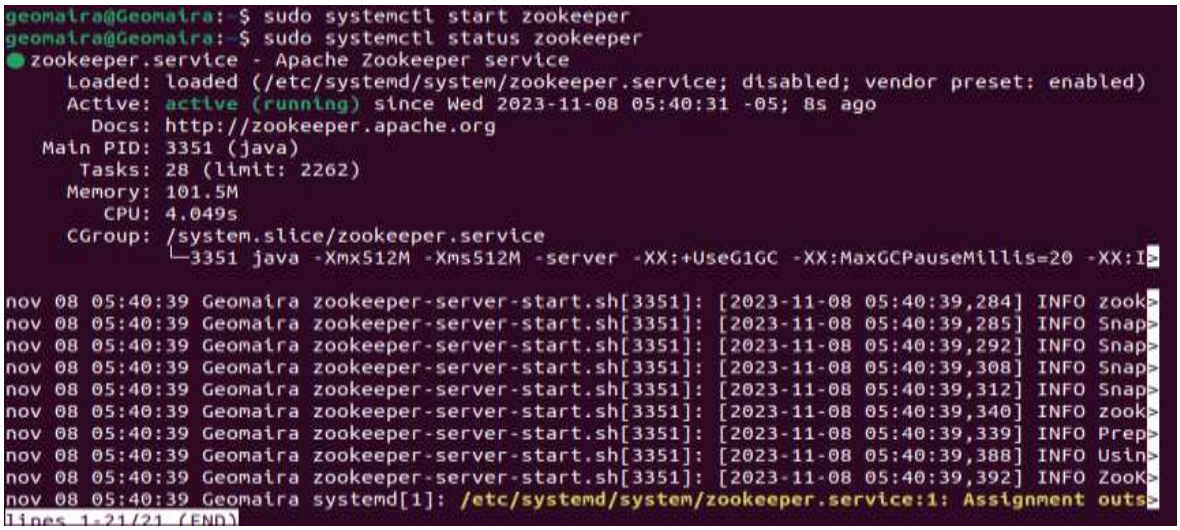

Figura 3.17 Inicia el servicio de Zookeeper

De la misma manera se realiza la inicialización del servicio de *Kafka* con el comando sudo systemctl start kafka como se ve en la Figura 3.18.

```
geomaira@Geomaira:~$ sudo systemctl start kafka
geomaira@Geomaira:~$ sudo systemctl status kafka
● kafka.service
     Loaded: loaded (/etc/systemd/system/kafka.service; disabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-11-08 05:44:41 -05; 11s ago
       Docs: http://kafka.apache.org/documentation.html
   Main PID: 3771 (java)
      Tasks: 71 (limit: 2262)
     Memory: 348.4M
        CPU: 7.559s
     CGroup: /system.slice/kafka.service
             └─3771 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -Xmx1G -Xms1G -server -XX:+UseG1G>

nov 08 05:44:49 Geomaira kafka-server-start.sh[3771]: [2023-11-08 05:44:49,228] INFO Awaiting soc>
nov 08 05:44:49 Geomaira kafka-server-start.sh[3771]: [2023-11-08 05:44:49,231] INFO [Controller >
nov 08 05:44:49 Geomaira kafka-server-start.sh[3771]: [2023-11-08 05:44:49,246] WARN [Controller >
nov 08 05:44:49 Geomaira kafka-server-start.sh[3771]: [2023-11-08 05:44:49,304] INFO [Controller >
nov 08 05:44:49 Geomaira kafka-server-start.sh[3771]: [2023-11-08 05:44:49,316] INFO Kafka versio>
nov 08 05:44:49 Geomaira kafka-server-start.sh[3771]: [2023-11-08 05:44:49,317] INFO Kafka commit>
```

Figura 3.18 Inicia el servicio de Kafka

Esquema diseño para la implementación del servicio de Kafka con DevOps

Cuando se verificó que los dos servicios se encuentran activos se realizó el diseño para el *playbook* que instalará el servicio de *Kafka* y *Zookeeper* en el cliente. A continuación, se muestra de la Figura 3.19. Se puede observar que se tiene a la máquina de tipo servidor de *Ansible* en la cual se albergan el inventario de *Host* que contiene a la dirección *IP* del cliente y por otro lado está el *playbook* que va a ejecutar las tareas para la instalación del servicio de *networking* en el cliente.

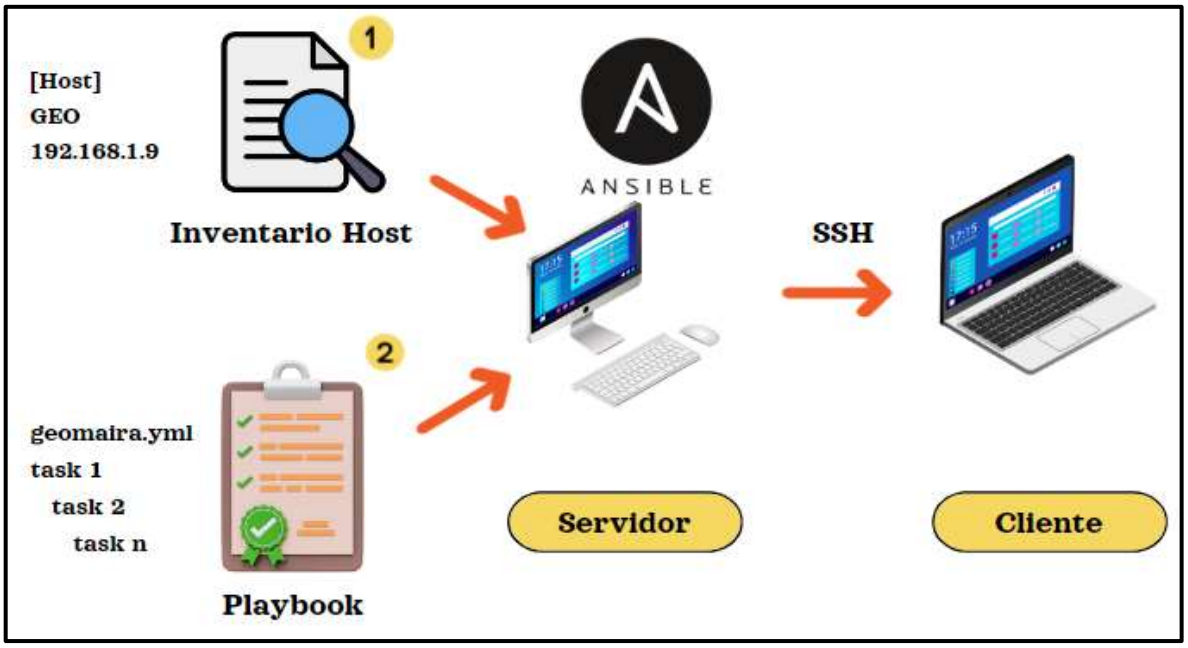

Figura 3.19 Diseño para la implementación de servicios de networking mediante DevOps

3.3 Implementar las soluciones mediante herramientas de DevOps para el despliegue de los servicios de networking

Finalizadas las pruebas de *Kafka* y *Zookeeper* en la máquina que actúa como servidor, se realiza la actualización de los repositorios usando el comando sudo-apt update como se evidencia en la Figura 3.20.

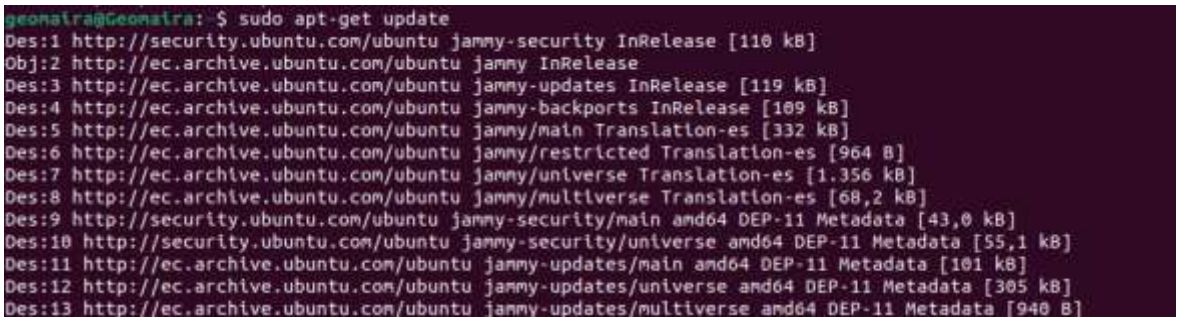

Figura 3.20 Actualización de repositorios

Como siguiente paso, se agrega el repositorio *PPA (Personal Package Archive)* que es el archivo de paquete personal correspondiente a una dependencia de *Ansible*. La cual permite agregar *software* diferente al repositorio *APT* y poder administrar como cualquier otra aplicación. Se instala mediante sudo apt install software-propties-common como se evidencia en la Figura 3.21. Pero también se tiene que actualizar el paquete instalado con el comando sudo apt upgrade software-propties-common como se ve en la Figura 3.22.

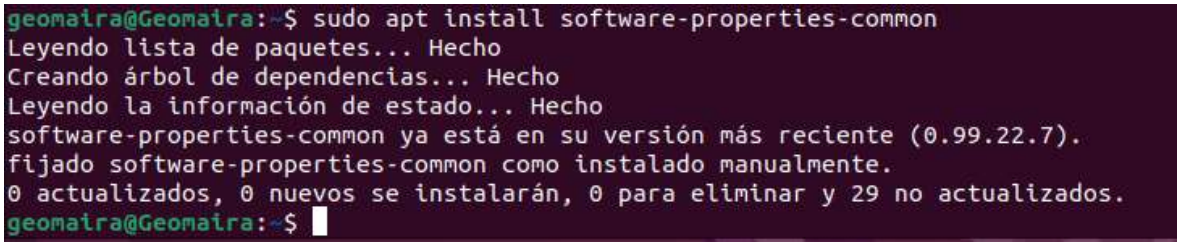

Figura 3.21 Instalación de repositorio PPA

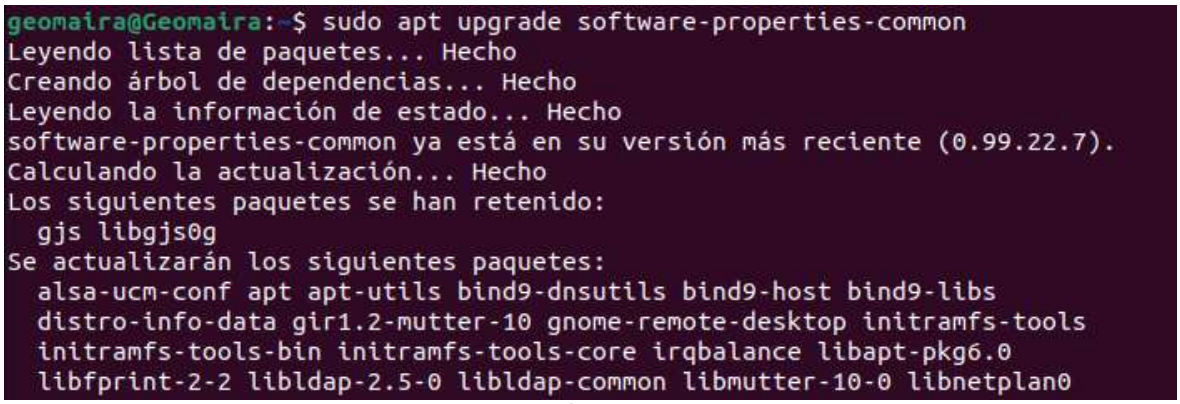

Figura 3.22 Actualización del paquete o repositorio

Cuando se añadió el repositorio anterior, *APT* va a permitir adquirir repositorios nuevos. En la Figura 3.23, se puede observar la agregación del repositorio para *Ansible* mediante sudo add-apt-repository –yes –update ppa:ansible/ansible.

```
geomaira@Geomaira:~$ sudo add-apt-repository --yes --update ppa:ansible/ansible
Repositorio: «deb https://ppa.launchpadcontent.net/ansible/ansible/ubuntu/ jammy
 main»
Descripción:
Ansible is a radically simple IT automation platform that makes your application
s and systems easier to deploy. Avoid writing scripts or custom code to deploy a
nd update your applications— automate in a language that approaches plain Englis
h, using SSH, with no agents to install on remote systems.

http://ansible.com/

If you face any issues while installing Ansible PPA, file an issue here:
https://github.com/ansible-community/ppa/issues
Más información: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Añadiendo repositorio.
```

Figura 3.23 Agregación del repositorio ppa:ansible/ansible

Al completar los pasos anteriores, se procede a la instalación de la última versión de Ansible mediante *APT,* usando el comando sudo apt install -y ansible en el modo de super usuario esto se evidencia en la Figura 3.24.

```
geomaira@Geomaira:~$ sudo apt install -y ansible
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  ansible
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 2 no actualizados.
Se necesita descargar 16,5 MB de archivos.
Se utilizarán 291 MB de espacio de disco adicional después de esta operación.
Des:1 https://ppa.launchpadcontent.net/ansible/ansible/ubuntu jammy/main amd64 a
nsible all 8.6.1-1ppa~jammy [16,5 MB]
Descargados 16,5 MB en 1min 3s (264 kB/s)
```

Figura 3.24 Instalación de Ansible

Un requisito para que el *software* funcione correctamente es que tiene que contar con *Python* el mismo debe encontrarse instalado en el sistema operativo, es por eso, que se va a instalar con el comando apt install python3-argcomplete como se evidencia en la Figura 3.25.

```
geomaira@Geomaira:~$ sudo apt install python3-argcomplete
[sudo] contraseña para geomaira:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
python3-argcomplete ya está en su versión más reciente (1.8.1-1.5).
Los paquetes indicados a continuación se instalaron de forma automática y ya no
son necesarios.
  linux-headers-6.2.0-36-generic linux-hwe-6.2-headers-6.2.0-36
```

Figura 3.25 Instalación de Python3

Una vez que se ha completado la instalación, se verifica que *Ansible* y *Python* se hayan instalado correctamente en el servidor. Mediante la línea de comando ansible –versión se observa en la Figura 3.26, la versión.

```
geomaira@Geomaira:~$ sudo ansible --version
[sudo] contraseña para geomaira:
ansible [core 2.15.6]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share
/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/co
llections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] (/usr/bin/
python3)
  jinja version = 3.0.3
  libyaml = True
```

Figura 3.26 Verificación de versión de Ansible

Conexión de SSH

Ansible hace el uso de un inventario de *hosts* el que se encuentra en la siguiente ruta /etc/ansible/hosts, para acceder a este archivo se usa el siguiente comando sudo nano /etc/ansible/hosts tal como se evidencia en la Figura 3.27. En este archivo se van a detallar los dispositivos que se van a controlar. Es importante la declaración del inventario ya que este empieza con una división de secciones mediante corchetes, a continuación, se indica el nombre con el que se identificará a los *hosts*, tal como se muestra en la Figura 3.28. Dentro de la sección también se declaran variables las cuales se usarán para el control de los dispositivos, estas son: contraseñas, nombres de usuarios, tipo de comunicación que utiliza, etc.

```
geomaira@Geomaira:~$ sudo nano /etc/ansible/hosts
```

Figura 3.27 Comando para acceder al archivo /etc/ansible/hosts

```
  GNU nano 6.2                    /etc/ansible/hosts

#HOSTS PARA EL CONTROL
[GEO1]
geo1 ansible_host=192.168.1.9
[GEO1:vars]
ansible_user=geo1
ansible_connection=ssh
ansible_password=geo1
```

Figura 3.28 Inventario de Hosts

Antes de continuar con la configuración del archivo ssh_config, se verifica que el servicio de *SSH* se encuentre activo y ejecutándose si no lo está se procederá a instalarlo con el comando sudo apt install openssh-server como se evidencia en la Figura 3.29.

Figura 3.29 Instalación del protocolo de SSH

A continuación, se accede al archivo sudo nano /etc/ssh/sshd_config en el cual se va a verificar que las siguientes líneas se encuentren habilitadas, en el caso de no estarlo se procede a habilitar como se observa en la Figura 3.30. Esto se verifica para que el servidor de *Ansible* permita la autenticidad a través de claves.

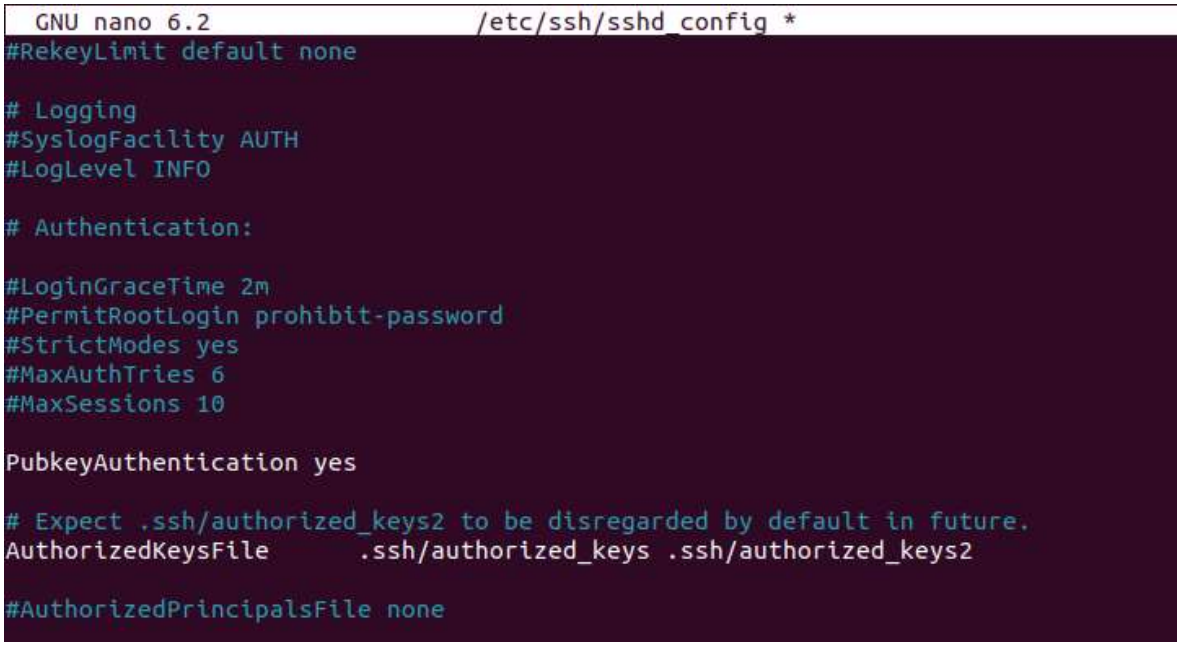

Figura 3.30 Verificación de parámetros activados en el archivo sshd_config

Siguiente a ello, se reinicia el servicio y se verifica su ejecución como lo muestra en la Figura 3.31.

Figura 3.31 Verificación de SSH activo y ejecutándose

Con la configuración realizada se continua con el siguiente paso en la Figura 3.32, se muestra el cambio que se realizó en los parámetros del archivo ssh_config la cual se encuentra la siguiente ruta /etc/shh/ssh_config, con la finalidad de lograr establecer una comunicación *SSH* sin problemas. En el archivo se eliminó el comentario de la siguiente línea ciphers aes128-ctr, aes192-ctr,aes256-crt,aes128-cbc,3des-cbc para que se detallen los algoritmos de cifrado que son seguros y que va a utilizar *Ansible* para mandar los datos por medio de la red. También, se añadió la siguiente línea KexAlgorithms +diffie-hellman-group1-sha1 para que permita establecer una comunicación segura entre dos nodos. El algoritmo de *Diffie-Hellman* admite la compartición de una clave la cual es secreta entre dos nodos.

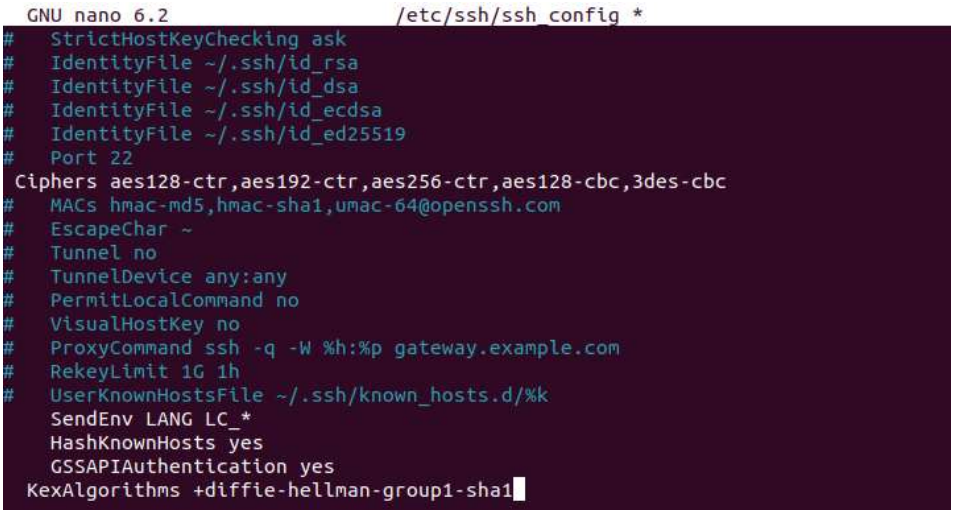

Figura 3.32 Edición del archivo SSH

En el cliente, es necesario que se habilite el protocolo *SSH* para permitir la comunicación con el servidor de *Ansible*. Por lo que se procedió a instalar el servicio como se observa en la Figura 3.33.

```
geo1@GEO1:~$ sudo apt install -y ssh
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  ncurses-term openssh-client openssh-server openssh-sftp-server ssh-import-id
Paquetes sugeridos:
  keychain libpam-ssh monkeysphere ssh-askpass molly-guard
Se instalarán los siguientes paquetes NUEVOS:
  ncurses-term openssh-server openssh-sftp-server ssh ssh-import-id
Se actualizarán los siguientes paquetes:
  openssh-client
1 actualizados, 5 nuevos se instalarán, 0 para eliminar y 130 no actualizados.
Se necesita descargar 755 kB/1.660 kB de archivos.
```

Figura 3.33 Instalación de SSH en el cliente

Se realiza la inicialización del servicio con el comando sudo systemctl start ssh.service y se verifica que se encuentre activo, como se muestra en la Figura 3.34.

```
geo1@GEO1:~$ sudo systemctl status ssh.service
[sudo] contraseña para geo1:
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: e>
     Active: active (running) since Wed 2024-02-14 03:15:23 -05; 5min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 1447 (sshd)
      Tasks: 1 (limit: 2262)
     Memory: 3.0M
        CPU: 99ms
     CGroup: /system.slice/ssh.service
             └─1447 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

feb 14 03:15:18 GEO1 systemd[1]: Starting OpenBSD Secure Shell server...
feb 14 03:15:23 GEO1 sshd[1447]: Server listening on 0.0.0.0 port 22.
feb 14 03:15:23 GEO1 sshd[1447]: Server listening on :: port 22.
```

Figura 3.34 Verificación del servicio SSH activado en el cliente

El nodo controlador necesita contar con las claves *SSH*, para generar dichas claves se lo hace mediante el comando ssh-keygen el cual nos arrojará un par de claves *RSA* por defecto como se evidencia en la Figura 3.35. El algoritmo *RSA* corresponde a un cifrado asimétrico el cual se encuentra basado en la dificultas de factorizar grandes números primos, garantizando así una seguridad alta al sistema. Estas claves que se generan se van a almacenar en el archivo denominado como id_rsa el mismo que se encuentra en la siguiente ruta /home/geomaira/.ssh/id_rsa. Como resultado se puede observar en la Figura 3.36.

Figura 3.35 Generación de claves de cifrado y descifrado

Figura 3.36 Verificación de archivos

Para concluir el proceso de la conexión de *SSH* del nodo controlador con el cliente, se realizó la copia de la clave pública de *SSH* que se generó del nodo controlador al cliente con la ejecución del comando ssh-copy id geo1@192.168.1.9.

En la Figura 3.37 se muestra cómo se ejecuta el comando que se mencionó anteriormente, y como resultado se muestra que se logró iniciar sesión en el dispositivo correctamente.

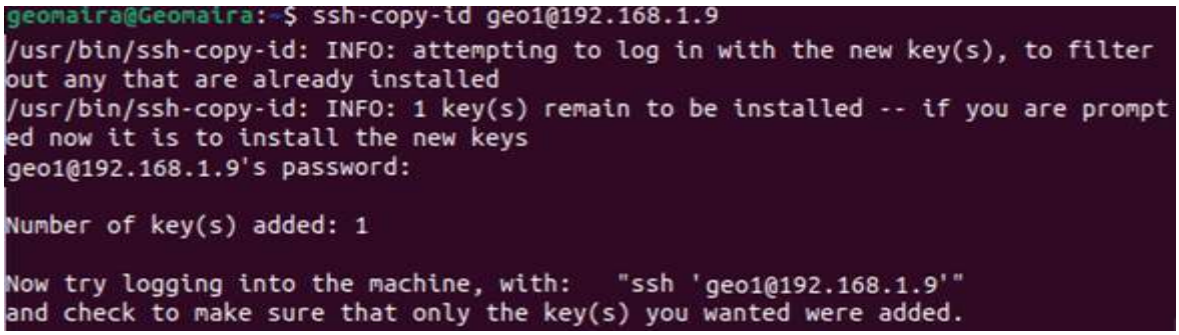

Figura 3.37 Conexión SSH añadida correctamente

El mensaje que se muestra en la figura anterior verifica que la conexión *SHH* fue exitosa pero sólo mediante la línea de comandos, pero esto no garantiza que se haya establecido la conexión remota entre los dispositivos, de manera, que se ejecuta el siguiente comando

ansible all -m ping all de la herramienta de *Ansible*, como se observa en la Figura 3.38.

En el contexto de *Ansible* se conoce que, si la conexión se estableció correctamente, cuando se envié un paquete de *ping* al *host* que es remoto, este va a dar una respuesta que se denomina como *pong*. Obteniendo así una respuesta a la solicitud enviada.

```
geomaira@Geomaira:~$ ansible all -m ping
geo1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Figura 3.38 Prueba de ping-pong

3.4 Implementación del playbook para el despliegue de kafka

Ansible utiliza *playbooks* para la ejecución de tareas, en donde para la creación del *playbook* se utiliza el comando sudo nano geomaira.yml. Se debe tener en cuenta que tiene un formato de tipo *YML*. Una vez dentro del archivo se empieza a realizar la edición de este. El *playbook* empieza con tres guiones al inicio del archivo "---", ya que esto especifica que está en formato *YAML*, en la siguiente línea se ubica el *name* que corresponde para describir al conjunto de tareas que se va a realizar. En cuanto al hosts se añade el nombre de la lista de *hosts* creados en el inventario. También se tiene *become* con su valor en *true* para que se ejecute la tarea en el modo de super usuario. Por otro lado, se tiene a gather_facts con su valor en *false* con la finalidad de que se obligue a que no se recopile información sobre el sistema del *host*, tal como se observa en la Figura 3.39.

```
  GNU nano 6.2                        geomaira.yml
---

#Prueba de PLAYBOOK KAFKA
- name: Primera prueba del playbook
  hosts: GEO1
  become: yes
  gather_facts: false
```

Figura 3.39 Líneas de configuración del playbook

En la Figura 3.40 se muestra las variables establecidas mediante el comando vars, en donde se establecieron las variables para instalar *Java* y *Kafka*. El uso de las llaves y guiones dobles hace referencia a una variable que se ha creado.

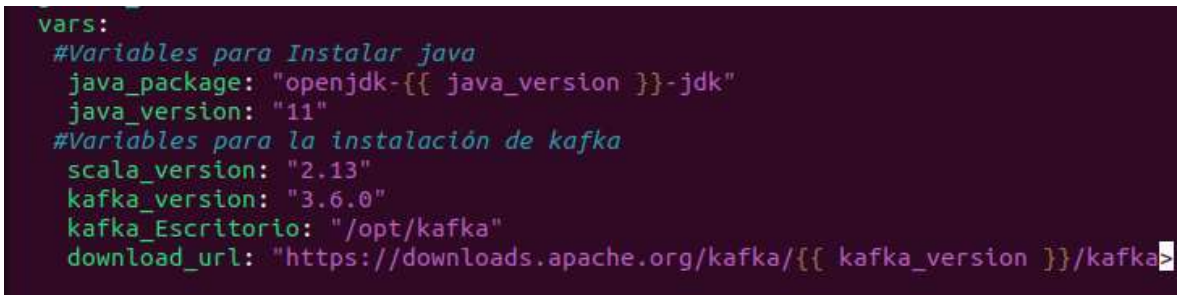

Figura 3.40 Variables establecidas

A continuación, se introduce el comando *tasks*, donde se definirán las tareas a realizar. Como se explicó anteriormente, se usa el comando *name* para identificar cada tarea, se debe tener en cuenta que la separación de los contenidos de cada sección tenga un formato que se encuentre estructurado.

La primera tarea asignada consiste en la actualización del repositorio, para lo cual se emplea el módulo ansible.builtin.apt. Dentro de este módulo, se especifica la primera tarea. Con "*" para que se actualicen todos los paquetes, también se define el *state* como *present* el que va a indicar que el paquete debe estar instalado. Además, se establece el parámetro Force_apt_get en *yes* para que *Ansible* forcé a apt-get para la gestión de los paquetes. Para el parámetro update_cache se configura en *true*, garantizando que *Ansible* primero actualice los paquetes antes de su ejecución, y cache_valid_time para que se actualice cada cierto tiempo, tal como se observa en la Figura 3.41.

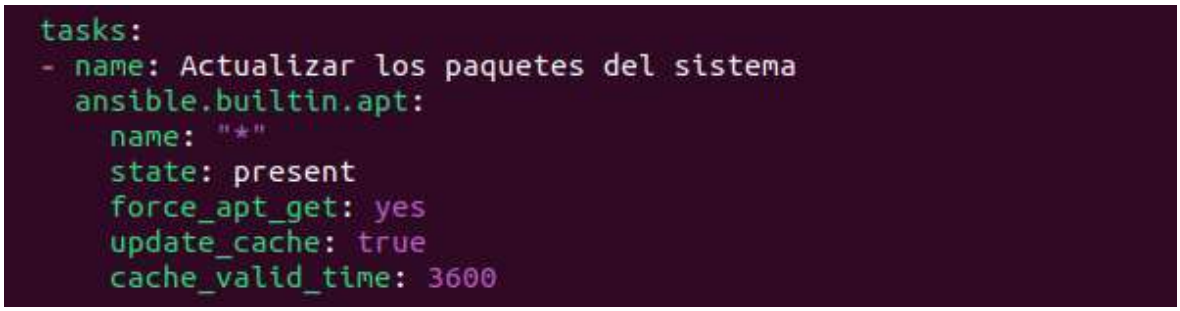

Figura 3.41 Actualizar Paquetes

La siguiente tarea corresponde a instalar *OpenJDK* 11 o *Java*, para lo cual se utilizó el módulo *apt* para la gestión de paquetes. Dentro de él se nombra el paquete a instalar y se hace el uso de la variable que se estableció, además, se establece el *state* en modo *present* esto se evidencia en la Figura 3.42.

```
- name: Instalación de OpenJDK 11
  apt:
    name: "{{ java_package }}"
    state: present
```

Figura 3.42 Instalar OpenJDK 11

En la Figura 3.43 se puede observar que se instaló correctamente el paquete de Java. Se establece la siguiente tarea utilizando el módulo *Shell* para que se ejecute java -version en donde se usa la siguiente instrucción 2>&1 que va a redirigir la salida. El comando *register* es donde se almacena el resultado de salida de la tarea en la variable java_version_output y por último ignore_errors en yes para que en el caso de que se produzca un error en el código así este haya sido exitoso, *Ansible* pueda seguir ejecutando las siguientes tareas.

```
- name: Verificación de la versión de Java instalado
  shell:  java -version 2>&1
  register: java_version_output
  ignore_errors: yes
```

Figura 3.43 Verificación de Java

Como siguiente tarea se tiene a la presentación de la versión de *Java* para lo cual se usa el módulo *debug* que permite que se impriman mensajes cuando se ejecute el *playbook*. Como se evidencia en la Figura 3.44.

```
- name: Salida de la versión de Java verificada
  debug:
    var: java_version_output.stdout_lines
```

Figura 3.44 Impresión de la versión de Java

En la Figura 3.45 se puede observar que la salida de la versión de *Java* se ejecutó correctamente.

```
TASK [Salida de la versión de Java verificada] ********************************
*
ok: [geo1] => {
    "java_version_output.stdout_lines": [
        "openjdk version \"11.0.21\" 2023-10-17",
        "OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu122.04
)",
```

Figura 3.45 Mensaje impreso de la versión de Java

Cuando se ha verificado que *Java* se instaló correctamente se procede a la siguiente tarea. Es la creación del directorio para el servicio de *Kafka*. Se utiliza el módulo ansible.builtin.file, ya que permite ver la ruta especifica de un archivo. Para crear el directorio se utiliza el comando *path* con el que se establece la ruta de archivo, se declaró en un principio

variables para la creación del directorio /opt/Kafka, es importante declarar que se establezca como directorio y otorgar los permisos necesarios con el comando *mode* y los valores de 0755, en donde el dígito “0” es para los permisos especiales en directorios, el siguiente dígito “7” indica que tiene permisos de escritura, lectura y ejecución, por último el dígito “5” que indica que sólo puede leer y ejecutar. En la Figura 3.46 se evidencia lo mencionado.

```
- name: Instalación de Apache kafka - Creación del directorio para Kafka
  ansible.builtin.file:
    path: "{{ kafka_Escritorio }}"
    state: directory
    mode: "0755"
```

Figura 3.46 Creación del directorio Kafka

Siguiente a ello, se tiene a la tarea para la descarga de *Kafka* la misma que se va a realizar mediante un *URL* por lo que se hace el uso del módulo get_url, su función es descargar archivos desde un URL para almacenarlos localmente. En donde se establece el comando *url* que es donde se va a indicar a la dirección, en un principio se estableció como variable download_url. Es importante determinar el destino donde se va a descargar el paquete, con la siguiente instrucción *dest* y se establece la ruta del directorio creado, sin olvidar que el paquete que se va a descargar se encuentra en binario por lo que se va a descargar en formato tgz. Se deben otorgar los permisos de leer y escribir, pero no de ejecutar, la Figura 3.47 lo demuestra.

```
- name: Descarga de Kafka mediante wget
  get_url:
    url: "{{ download_url }}"
    dest: "{{ kafka_Escritorio }}/kafka.tgz"
    mode: '0644'
  register: download_result
```

Figura 3.47 Descarga de Kafka

Luego en la Figura 3.48 se observa la siguiente tarea que corresponde a descomprimir el archivo que se descargó. Para esto se utiliza el módulo ansible.builting.unarchive en donde se van a establecer los siguientes parámetros. El *src* es utilizado para especificar la ruta de ubicación del archivo de origen indicando en que formato se encuentra, después se define la ruta de destino para el archivo copiado.

```
- name: Descomprimir el archivo comprimido kafka
  ansible.builtin.unarchive:
    src: "{{ kafka_Escritorio }}/kafka.tgz"
    dest: "{{ kafka_Escritorio }}"
    remote_src: true
  register: unarchive_result
```

Figura 3.48 Descomprimir el archivo

La siguiente tarea corresponde a la creación del archivo *systemd* para el servicio de *Zookeeper*. Se hace el uso del módulo ansible.builtin.copy ya que permite copiar un archivo de una ubicación raíz a otra ubicación destino. Se establecen los siguientes parámetros como *dest* en el que se ubica la ruta del archivo *systemd* para *zookeeper*. Siguiente a ello, el parámetro *content* da paso a insertar líneas de texto en el archivo *systemd*, para que se edite dicho archivo se inicia con el siguiente símbolo "|".

El archivo *systemd* nos sirve para gestionar el servicio de *Zookeeper*. El cual inicia indicando con la ruta del archivo que se creó para *Zookeeper*. A continuación, se tiene la línea [*Unit*] que contiene las características con las dependencias para la unidad.

- *Description*, en la cual se añade el nombre del servicio.
- *Documentation*, se especifica una *URL* acerca de la documentación del servicio.
- *Requires*, en esta línea se especifica las dependencias para que sea ejecutado el archivo como por ejemplo primero tiene que encontrarse el dispositivo en red y los sistemas de archivos remotos se hayan inicializado correctamente.
- *After*, establece el orden como se va a iniciar las unidades con respecto a otras unidades.

En la Figura 3.49 se evidencia lo antes mencionado.

```
- name: Creación de archivo systemd para Servicio Zookeeper
  ansible.builtin.copy:
    dest: /etc/systemd/system/zookeeper.service
    content: |
      /etc/systemd/system/zookeeper.service
      [Unit]
      Description=Apache Zookeeper service
      Documentation=http://zookeeper.apache.org
      Requires=network.target remote-fs.target
      After=network.target remote-fs.target
```

Figura 3.49 Parámetros para el archivo del servicio de Zookeeper

A continuación, en la Figura 3.50 se tiene la línea de [Service] en el que se va a definir como *systemd* va a gestionar el servicio de *Zookeeper* cuando ya se haya iniciado. Se muestran las líneas a ser editadas.

- *Type*, se especifica que tipo de proceso que se va a iniciar en el servicio en este caso es simple, es decir, de manera sencilla.
- *ExecStart*, se indica a *systemd* que va a iniciar el servicio al ejecutar el *script* de zookeeper-server-start.sh el mismo que se encuentra en la ruta /opt/kafka/bin/.

- *ExecStop*, se indica a *systemd* que va a detener el servicio al ejecutar el *script* de zookeeper-server-stop.sh que se encuentra en la ruta /opt/kafka/bin.
- *Restart*, se especifica a *systemd* cuando debe reiniciar el servicio en este caso cuando sea on- abnormal, es decir, cuando el código tenga una salida distinta a cero. Con esto se garantiza que el servicio se encuentre activo en el caso de problemas inesperados.

```
[Service]
Type=simple
ExecStart=/opt/kafka/kafka_2.13-3.6.0/bin/zookeeper-server-start.sh /opt/kafka/kafka_2.13-3.6.0/config/zookeeper.properties
ExecStop=/opt/kafka/kafka_2.13-3.6.0/bin/zookeeper-server-stop.sh
Restart=on-abnormal
```

Figura 3.50 Gestión del servico Zookeeper

Por último, la línea [*Install*] en el archivo de unidad *systemd* indica como la unidad debe ser habilitada para que su arranque sea de manera automática. Se estableció el parámetro *WantedBy*, este establece que el servicio sea habilitado cuando se encuentre en modo multiusuario. Este modo significa que el sistema se encuentra operativo y varios usuarios pueden acceder a él. Para finalizar esta tarea se añaden los permisos para poder acceder a este archivo de *systemd* de *Zookeeper*, como se observa en la Figura 3.51.

```
  [Install]
  WantedBy=multi-user.target
mode: "0755"
```

Figura 3.51 Habilitación de la unidad systemd

En la siguiente tarea se va a crear el archivo de *systemd* para el servicio de *Kafka*. Se sigue el mismo proceso mencionado anteriormente con la diferencia que cambia el nombre de la ruta de la ubicación del archivo, en este caso es /etc/systemd/system/Kafka.service.

En la Figura 3.52 se verifica las líneas que contiene el archivo *systemd* de *Kafka*, cambian los siguientes parámetros. En la línea de [*Unit*] que contiene las características con las dependencias para la unidad.

- *Documentation*, se especifica una *URL* acerca de la documentación del servicio.
- *Requires*, en esta línea se especifica las dependencias para el servicio de *Kafka*, es decir, que *Zookeeper* debe estar activo y ejecutándose para que el servicio actual se active correctamente.

```
- name: Creación de archivo Systemd para Servicio Kafka
  ansible.builtin.copy:
    dest: /etc/systemd/system/kafka.service
    content: |
      [Unit]
      Description=Apache Kafka Service
      Documentation=http://kafka.apache.org/documentation.html
      Requires=zookeeper.service
```

Figura 3.52 Parámetros para el servicio de Kafka

A continuación, en la Figura 3.53 se tiene la línea de [*Service*] en el que se va a definir como *systemd* va a gestionar el servicio de Kafka cuando ya se haya iniciado. Se muestran las líneas que cambian.

- *Environmentt*, establece a la variable Java_Home que es en donde se instaló Java, ya que es importante para la ejecución de *Kafka*.
- *ExecStart*, se indica a *systemd* que va a iniciar el servicio al ejecutar el script de kafka-server-start.sh el mismo que se encuentra en la ruta /opt/kafka/Kafka_2.13-3.6.0/bin/.
- *ExecStop*, se indica a *systemd* que va a detener el servicio al ejecutar el *script* de kafka-server-stop.sh que se encuentra en la ruta /opt/kafka/Kafka_2.13-3.6.0/bin/.

```
[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"
ExecStart=/opt/kafka/kafka_2.13-3.6.0/bin/kafka-server-start.sh /opt/kafka/kafka_2.13-3.6.0/config/server.properties
ExecStop=/opt/kafka/kafka_2.13-3.6.0/bin/kafka-server-stop.sh
```

Figura 3.53 Gestión del servicio Kafka

En la siguiente línea se tiene [*Install*] en el archivo *systemd* que indica que la unidad está habilitada para su arranque automático. De igual manera se estableció el parámetro *WantedBy*, este establece que el servicio sea habilitado cuando se encuentre en modo multiusuario. Para finalizar esta tarea se añaden los permisos para poder acceder a este archivo de *systemd* de *Kafka*, como se observa en la Figura 3.54.

```
    [Install]
    WantedBy=multi-user.target
  mode: "0755"
```

Figura 3.54 Habilitación de la unidad systemd

En la Figura 3.55 , se procede a realizar la inicialización de los servicios tanto de *Zookeper* como *Kafka*. Para lo cual se utilizó el módulo ansible.builtin.systemd que su principal función es automatizar a la gestión de servicios, así como también a las unidades de *systemd*. En donde se declaró también daemon-reload en el valor de yes, esto para que se reinicie el *Daemon*.

```
- name: Iniciar Servicio de Zookeeper
  ansible.builtin.systemd:
    name: zookeeper
    state: started
    daemon-reload: yes
    enabled: yes

- name: Iniciar Servicio de kafka
  ansible.builtin.systemd:
    name: kafka
    state: started
    daemon-reload: yes
    enabled: yes
```

Figura 3.55 Inicialización de servicios

Ejecución del playbook

Para ejecutar el *playbook* que se encuentra en el nodo controlador de *Ansible*, se lo realiza mediante el comando ansible-playbook geomaira.yml –ask-become-pass. En donde se va a mostrar lo siguiente *BECOME password* en el cual nos pedirá ingresar la contraseña del cliente. Cuando se autentifique empezará a desplegarse las tareas. Si la ejecución es exitosa se muestra un *OK* y el nombre del *host*, pero también se muestra un *changed* que indica que se realizó un cambio en el sistema cuando se instaló Java por lo tanto se ejecutó correctamente como se muestra en la Figura 3.56.

```
geomaira@Geomaira: ~
geomaira@Geomaira:~$ ansible-playbook geomaira.yml --ask-become-pass
BECOME password:

PLAY [Primera prueba del playbook] ********************************************

TASK [Actualizar los paquetes del sistema] ************************************
ok: [geo1]

TASK [Instalación de OpenJDK 11] **********************************************
ok: [geo1]

TASK [Verificación de la versión de Java instalado] ***************************
changed: [geo1]

TASK [Salida de la versión de Java verificada] ********************************
ok: [geo1] => {
    "java_version_output.stdout_lines": [
        "openjdk version \"11.0.21\" 2023-10-17",
        "OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu122.04)
",
        "OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Ubuntu-0ubuntu122.04, mi
xed mode, sharing)"
    ]
}
```

Figura 3.56 Ejecución de playbook

En la Figura 3.57, empieza la instalación del servicio de *Kafka* y *Zookeeper*. Como se evidencia al finalizar la ejecución se muestra el número de tareas que se ejecutaron.

Figura 3.57 Playbook Finalizado

3.5 Verificar el funcionamiento de cada servicio de networking implementado mediante DevOps

Ping al servidor

En la Figura 3.58, se realizó la verificación de conexión desde el cliente geo1 al servidor denominado como geomaira, usando el comando *ping* y la dirección del servidor que en este caso corresponde a la siguiente dirección IP 192.168.1.11. Se observa que los paquetes enviados desde nuestro cliente al servidor llegan sin inconvenientes y no existen perdidas de estos.

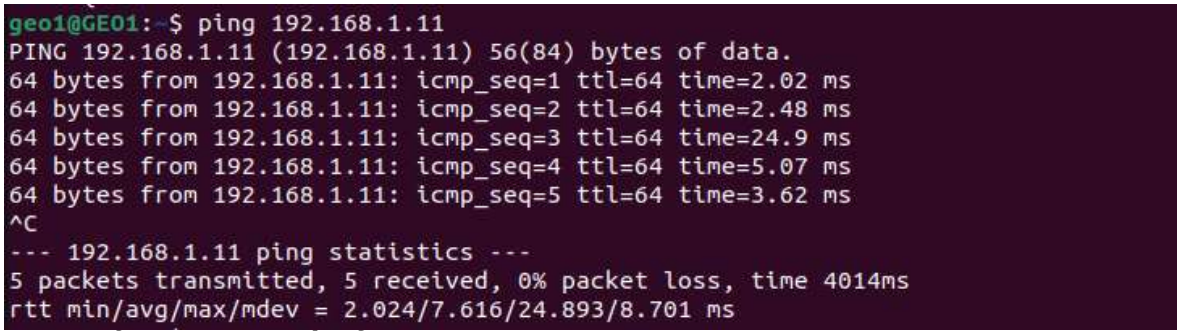

Figura 3.58 Ping Cliente- Servidor

De igual manera, se realizó la prueba de *ping* desde nuestro servidor al cliente, que tiene una dirección IP 192.168.1.9. Teniendo como resultado que no existen perdidas de paquetes tal como se muestra en la Figura 3.59. Se observa que se estableció correctamente la conexión *SSH* de la máquina cliente al servidor.

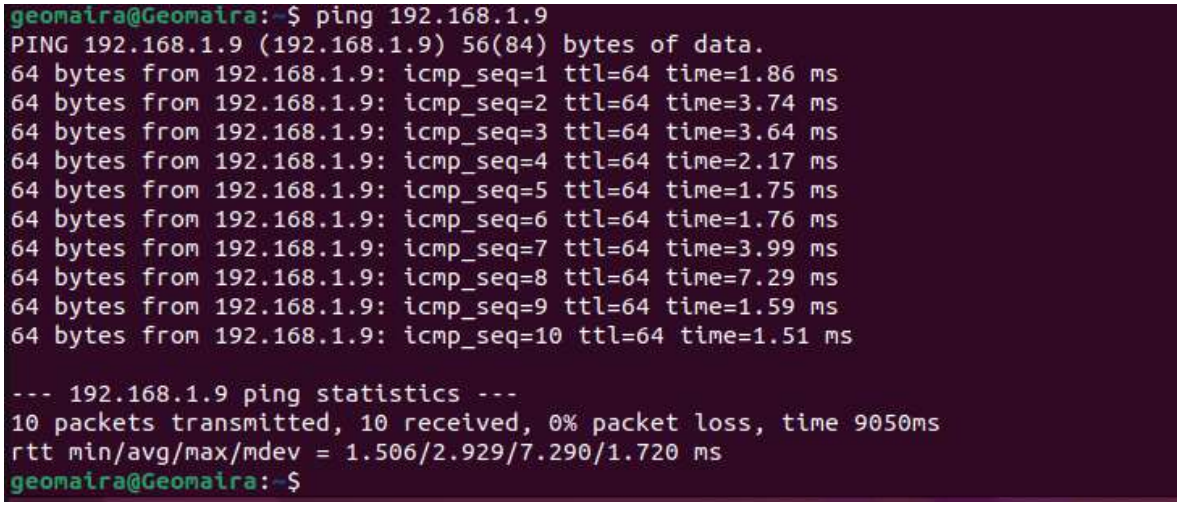

Figura 3.59 Ping Servidor- Cliente

Determinando así que al no haber pérdida de paquetes en las pruebas realizadas con el *ping* tanto del cliente a servidor y viceversa, se establece que la conexión se encuentra estable.

Conexión SSH

Cuando se ha verificado que tenemos comunicación desde el cliente al servidor y viceversa, se procede a realizar las pruebas de conexión de *SSH* desde el cliente al servidor. A continuación, en la Figura 3.60, mediante el comando sudo ssh nombre del servidor@dirección IP de este. En este caso nuestro servidor es geomaira@192.168.1.11, nos pedirá la contraseña de nuestro dispositivo cuando se haya verificado nos pedirá ingresar la contraseña del servidor, al ser autenticado correctamente se establece una sesión segura en el intercambio de datos y como se observa ya nos encontramos dentro del servidor.

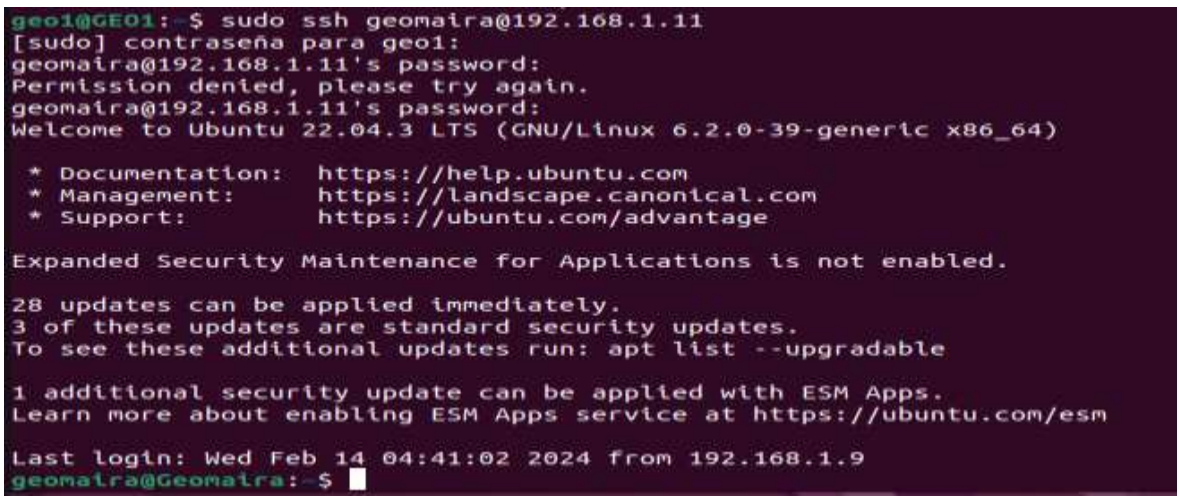

Figura 3.60 Conexión establecida SSH - Cliente al servidor

Se realiza el mismo proceso para la verificación de servidor al cliente, como se observa en la Figura 3.61. Se verifica que se estableció una conexión exitosa y se ingresó al cliente.

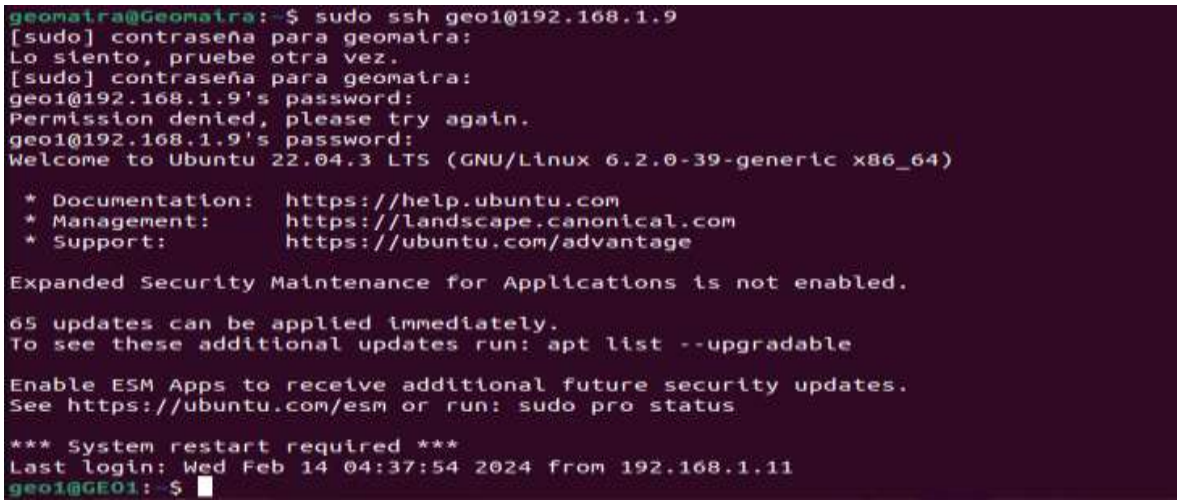

Figura 3.61 Conexión establecida SSH - Servidor a Cliente

Servicio Kafka y Zookeeper

Ingresamos a la máquina denominada como cliente geo1, en el cual se accede al terminal y se verifica que procesos de *Java* que se encuentren ejecutando con el siguiente comando sudo jps. Se logra observar en la Figura 3.62 que se encuentra ejecutándose el servicio de *Kafka* y *Java* con el nombre de *QuorumPeerMain* cada uno con su *ID* del proceso.

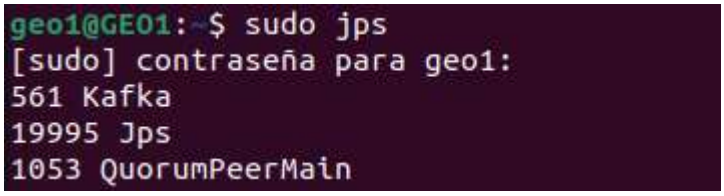

Figura 3.62 Procesos de Java ejecutándose en el cliente

Cuando se verificó que se está ejecutando *Kafka*, se procede a verificar accediendo al directorio creado con el comando cd /opt/Kafka, siguiente a ellos se utiliza el comando ls para que nos muestre los archivos que se encuentran en ese directorio. Mostrándose el archivo que se descargado en formato .tgz y el archivo descomprimido, tal como se observa en la Figura 3.63.

Figura 3.63 Directorio Kafka

Ahora en la Figura 3.64, se ingresa al archivo descomprimido y se verifica los archivos ocultos que contiene tales como el archivo bin, licencia, el archivo de config y etc.

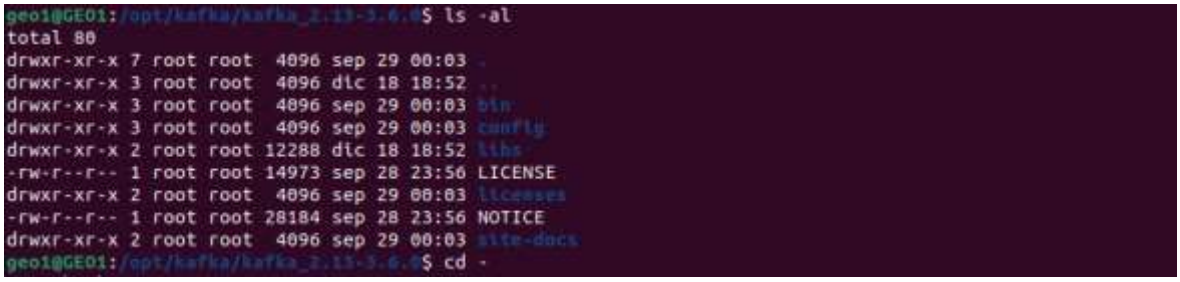

Figura 3.64 Verificación del contenido del archivo descomprimido

Finalmente se verifica que el servicio de *Zookeeper* y *Kafka* se encuentren activados con el siguiente comando sudo systemctl status Kafka y sudo systemctl status zookeeper, como se muestran en las Figura 3.65 y Figura 3.66.

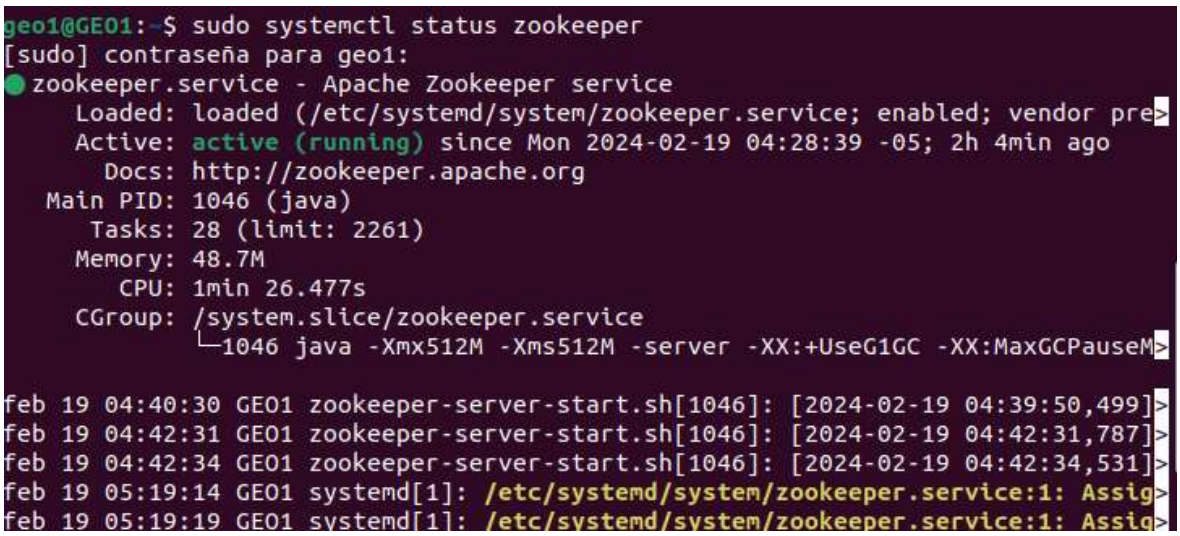

Figura 3.65 Servicio de Zookeeper - Activado

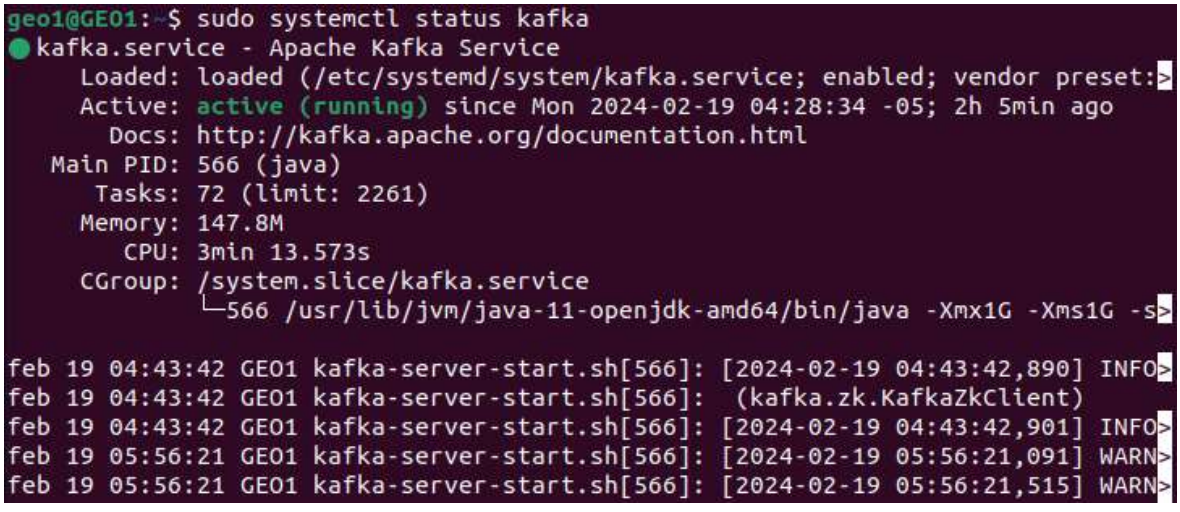

Figura 3.66 Servicio de Kafka – Activado

4 CONCLUSIONES

- Los servicios de *DevOps* se centran en proporcionar velocidad, eficiencia y seguridad en entornos de *networking*, ya que tienen la capacidad de adaptarse a las necesidades específicas de cada organización. Esto posibilita la gestión de redes de manera confiable y garantiza la flexibilidad necesaria para responder a los desafíos cambiantes del entorno empresarial.
- Las prácticas, herramientas y principios de *DevOps* convergen para convertirlo en un recurso altamente eficiente, capaz de ofrecer respuestas rápidas y efectivas. Este enfoque no solo garantiza tiempos de entrega reducidos, sino que también fomenta una retroalimentación constante que impulsa la mejora continua en todos los aspectos del proceso de desarrollo y operaciones.
- Ansible es una poderosa herramienta de automatización que facilita la ejecución de tareas para la configuración, gestión de infraestructura e implementación. Su enfoque en la infraestructura como código permite simplificar los procesos, lo que resulta en una administración más eficiente y escalable de los entornos de *TI*.
- *Kafka* desempeña funciones esenciales al permitir la publicación y suscripción de flujos de datos, procesarlos en tiempo real y almacenar los registros en el orden exacto en que se producen. Estas capacidades hacen de *Kafka* una herramienta fundamental para el procesamiento y la gestión de datos en tiempo real.
- Gracias a su escalabilidad, Ansible puede adaptarse fácilmente a diversos entornos. Su arquitectura modular le permite gestionar múltiples nodos simultáneamente, lo que lo convierte en una opción altamente flexible para implementaciones a gran escala en servicios de *networking*.
- Para desplegar la *playbook* destinada a implementar el servicio de *Kafka*, fue necesario emplear módulos específicos, ejecutar comandos precisos y seguir una guía detallada para asegurar una ejecución sin errores. Este enfoque meticuloso garantiza una implementación exitosa del servicio y minimiza posibles fallos durante el proceso.
- El protocolo *SSH* fue fundamental para la instalación de *Ansible*, dado que proporciona una comunicación segura y cifrada. Esto quedó evidenciado durante la copia de la clave *SSH*, que permitió establecer una comunicación segura con el cliente host, garantizando la integridad y confidencialidad de los datos durante el proceso de instalación.

- Se constató que la administración del dispositivo final es factible a través de Ansible, pero requiere un proceso secuencial. Este proceso comienza con la generación de las claves *SSH*, las cuales son compartidas con el host a administrar para la autenticación. Esto permite la ejecución de los *playbooks*, los cuales contienen las tareas específicas destinadas al dispositivo final.
- Las pruebas de verificación se llevaron a cabo utilizando el *playbook* denominado "geomaira.yml". Durante este proceso, se confirmó que tanto el servicio de *Kafka* como el de *Zookeeper* estaban activos y en funcionamiento en el dispositivo final.

5 RECOMENDACIONES

- Cuando se hayan creado las máquinas virtuales se debe poner direcciones IP fijas.
- Se recomienda instalar previamente y asegurarse de que los servicios de SSH estén activos tanto en el servidor como en el cliente antes de configurar la conexión *SSH* entre ambas máquinas.
- Para la configuración de *SSH* se debe cambiar ciertos parámetros en el archivo de config.ssh para que permita realizar el intercambio de claves sin errores. Dependiendo del caso.
- Es importante especificar correctamente en el archivo de *hosts* o inventario el medio a través del cual se va a comunicar el nodo controlador con el *host*.
- Dentro del *playbook* es necesario utilizar los módulos y comandos que soporte Ansible, ya que si no es el caso se va a producir un error y no se ejecutará la tarea. Además, si se usa “&” se debe utilizar el módulo *Shell* y para la edición de archivos *command*, ya que *Ansible* puede tomar como valores durante la ejecución.
- Al crear los archivos de *systemd* para *Zookeeper* y *Kafka*, es fundamental especificar correctamente las rutas del directorio donde se encuentran y dónde se almacenarán. Si estas rutas son incorrectas, ninguno de los dos servicios se ejecutará correctamente.
- Para ejecutar el *playbook* se debe agregar el comando de *pass* para que permita ingresar una contraseña para la ejecución en el cliente.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] NetApp, «NetApp,» [En línea]. Available: https://www.netapp.com/es/devops-solutions/what-is-devops/#:~:text=DevOps%20es%20un%20marco%20de,o%20productos%20para%20los%20clientes. [Último acceso: 25 Diciembre 2023].

[2] R. Hat, «Red Hat,» Diciembre 2024. [En línea]. Available: https://www.redhat.com/en/topics/automation/learning-ansible-tutorial. [Último acceso: 26 Diciembre 2023].

[3] M. Noufal, «Liquid Web,» 24 Mayo 2023. [En línea]. Available: https://www.liquidweb.com/kb/what-is-ansible/. [Último acceso: 25 Diciembre 2023].

[4] KeepCoding, «KeepCoding,» 3 Febrero 2023. [En línea]. Available: https://keepcoding.io/blog/que-es-y-como-crear-un-playbook-en-ansible/. [Último acceso: 26 Diciembre 2023].

[5] C. BasuMallick, «Spiceworks,» 28 Julio 2022. [En línea]. Available: https://www.spiceworks.com/tech/data-management/articles/what-is-kafka/. [Último acceso: 27 Diciembre 2023].

[6] C. Fernandez, «Sumo Logic,» 24 Agosto 2022. [En línea]. Available: https://www.sumologic.com/blog/devops-automation-best-practices-benefits/. [Último acceso: 27 Diciembre 2023].

[7] C. Victor, «SumoLogic,» 24 Marzo 2019. [En línea]. Available: https://arquitectoit.com/devops/7-beneficios-devops/. [Último acceso: 10 Febrero 2023].

[8] S. Manjaly, «BLOG,» 23 Marzo 2023. [En línea]. Available: https://blog.invgate.com/es/ansible. [Último acceso: 17 Febrero 2024].

[9] D. Subramanian, «Builtin,» 17 Enero 2024. [En línea]. Available: https://builtin.com/data-science/what-is-kafka. [Último acceso: 25 Diciembre 2023].

7 ANEXOS

ANEXO I: Certificado de Originalidad

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 21 de febrero de 2024

De mi consideración:

Yo, FERNANDO BECERRA, en calidad de Director del Trabajo de Integración Curricular titulado IMPLEMENTACIÓN DE KAFKA MEDIANTE DEVOPS elaborado por el estudiante GEOMAIRA PASTO de la carrera en TECNÓLOGÍA SUPERIOR EN REDES Y TELECOMUNICACIONES, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 12%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el link del informe generado por la herramienta Turnitin.

LINK

https://epnecuador-my.sharepoint.com/:f:/g/personal/fernando_becerrac_epn_edu_ec/ElGlW7dMzSdNqmWUl9di9vsBSgYOjgEPOS_UFw8-nJX6YQ?e=YRHxg2

Atentamente,

Fernando Vinicio Becerra Camacho

Docente

Escuela de Formación de Tecnólogos

ANEXO II: Enlaces

Anexo II.I Código QR de la implementación y pruebas de funcionamiento

ANEXO III: Códigos Fuente

Playbook

```
---
#Prueba de PLAYBOOK KAFKA
- name: Primera prueba del playbook
  hosts: GEO1
  become: yes
  gather_facts: false
  vars:
   #Variables para Instalar java
    java_package: "openjdk-{{ java_version }}-jdk"
    java_version: "11"
   #Variables para la instalación de kafka
    scala_version: "2.13"
    kafka_version: "3.6.0"
    kafka_Escritorio: "/opt/kafka"
    download_url: "https://downloads.apache.org/kafka/{{ kafka_version }}/kafka>
tasks:
  - name: Actualizar los paquetes del sistema
    ansible.builtin.apt:
      name: "*"
      state: present
      force_apt_get: yes
      update_cache: true
```

```
    cache_valid_time: 3600
- name: Instalación de OpenJDK 11
  apt:
    name: "{{ java_package }}"
    state: present
- name: Verificación de la versión de Java instalado
  shell:  java -version 2>&1
  register: java_version_output
  ignore_errors: yes
- name: Salida de la versión de Java verificada
  debug:
    var: java_version_output.stdout_lines
- name: Instalación de Apache kafka - Creación del directorio para Kafka
  ansible.builtin.file:
    path: "{{ kafka_Escritorio }}"
    state: directory
    mode: "0755"
- name: Descarga de Kafka mediante wget
  get_url:
    url: "{{ download_url }}"
    dest: "{{ kafka_Escritorio }}/kafka.tgz"
    mode: '0644'
  register: download_result
- name: Descomprimir el archivo comprimido kafka
  ansible.builtin.unarchive:
```

```
    src: "{{ kafka_Escritorio }}/kafka.tgz"
    dest: "{{ kafka_Escritorio }}"
    remote_src: true
  register: unarchive_result
- name: Creación de archivo systemd para Servicio Zookeeper
  ansible.builtin.copy:
    dest: /etc/systemd/system/zookeeper.service
    content: |
      /etc/systemd/system/zookeeper.service
      [Unit]
      Description=Apache Zookeeper service
      Documentation=http://zookeeper.apache.org
      Requires=network.target remote-fs.target
      After=network.target remote-fs.target
      [Service]
      Type=simple
      ExecStart=/opt/kafka/kafka_2.13-3.6.0/bin/zookeeper-server-start.sh /op>
      ExecStop=/opt/kafka/kafka_2.13-3.6.0/bin/zookeeper-server-stop.sh
      Restart=on-abnormal
      [Install]
      WantedBy=multi-user.target
    mode: "0755"
- name: Creación de archivo Systemd para Servicio Kafka
  ansible.builtin.copy:
    dest: /etc/systemd/system/kafka.service
```

```
    content: |
      [Unit]
      Description=Apache Kafka Service
      Documentation=http://kafka.apache.org/documentation.html
      Requires=zookeeper.service
      [Service]
      Type=simple
      Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"
      ExecStart=/opt/kafka/kafka_2.13-3.6.0/bin/kafka-server-start.sh /opt/ka>
      ExecStop=/opt/kafka/kafka_2.13-3.6.0/bin/kafka-server-stop.sh
      [Install]
      WantedBy=multi-user.target
    mode: "0755"
- name: Iniciar Servicio de Zookeeper
  ansible.builtin.systemd:
    name: zookeeper
    state: started
    daemon-reload: yes
    enabled: yes
- name: Iniciar Servicio de kafka
  ansible.builtin.systemd:
    name: kafka
    state: started
    daemon-reload: yes
    enabled: yes
```