UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Análisis de funcionalidad de Zookeeper.

TRABAJO DE TITULACIÓN.

AUTOR: Samaniego Asanza, Ximena Lisseth.

DIRECTOR: Torres Tandazo, Rommel Vicente, PhD.

LOJA - ECUADOR

2017

2017

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Ingeniero.
Rommel Vicente Torres Tandazo.
DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de fin de titulación: **Análisis de funcionalidad de Zookeeper,** realizado por Samaniego Asanza Ximena Lisseth, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, Septiembre de 2017

f) . . . . . .

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

"Yo Samaniego Asanza Ximena Lisseth declaro ser autor (a) del presente trabajo de fin de titulación: **Análisis de funcionalidad de Zookeeper**, de la Titulación Sistemas Informáticos y Computación, siendo Rommel Torres director (a) del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 67 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado que se realicen a través, o con el apoyo financiero, académico o institucional (operativo) de la Universidad”

f.....

Autor Samaniego Asanza Ximena Lisseth.

Cédula 0706417524

DEDICATORIA

El presente trabajo va dedicado principalmente a Dios, ya que gracias a el he logrado concluir mi carrera.

A mis queridos padres Jorge Samaniego y Deysi Asanza, porque ellos siempre estuvieron a mi lado brindándome su apoyo y sus consejos para hacer de mí una mejor persona.

A mi hermana Ana Paula y mis abuelitos paternos y maternos por sus palabras, a mis abuelitos paternos que desde el cielo siempre me cuidan y me guían para que todo salga bien.

A mi esposo por sus palabras y confianza, por su amor y brindarme el tiempo necesario para realizarme profesionalmente, a mis amigos, compañeros y todos aquellas personas que de una u otro manera han contribuido para el logro de mis objetivos.

A mi amada hija Kristel por ser mi fuente de motivación e inspiración para poder superarme cada día más y así poder luchar para que la vida nos depare un futuro mejor.

A toda mi familia quienes han sido un motivo de inspiración, motivación y apoyo en mi proceso académico.

A todos ellos, infinitas Gracias porque fueron mi principal fuente de inspiración para poder cumplir este logro tan importante en mi vida.

AGRADECIMIENTO

Agradezco a Dios por haberme permitido llegar, a esta etapa de mi vida, y por tantas bendiciones otorgadas.

A la UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA, por brindarme la oportunidad de estudiar y cumplir con este logro importante en mi vida profesional.

A todos los docentes del área de Sistemas que fueron parte de mi formación académica universitaria, por sus experiencias y conocimientos que permitieron formarme como profesional.

Agradezco especialmente a mi director del trabajo de fin de titulación PhD, PMP. Rommel Torres, por su apoyo, conocimiento, así como también tenido toda la paciencia del mundo para guiarme durante todo el desarrollo de la tesis.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo de titulación describe los procedimientos de una herramienta Distribuido llamado Zookeeper, desde la instalación, configuración y funcionamiento, los mismos que se realizan con el fin de administrar, soporte a grandes cantidades de información, se realizó la selección de una aplicación que resulto de los requerimientos de la investigación Distribuida que cuente con base de datos NoSQL, Asimismo se hizo un análisis de Apache Zookeeper donde se comprendió el servicio que ofrece Zookeeper como: Consenso, Elección del líder y Creación/eliminación de nodos. Sin embargo, se plasmó con el fin de administrar procesos que utiliza tanto el servicio de Zookeeper y Cassandra. Y finalmente con la herramienta DataDog se pudo monitorizar algunas métricas como: rendimiento del equipo y funcionamiento del sistema Distribuido Zookeeper y Cassandra con el propósito de recomendar el sistema idóneo para uso de entornos Distribuidos.

PALABRAS CLAVES: NoSQL, sistemas Distribuidos, procesos Distribuidos, consenso.

ABSTRACT

The present titling work describes the procedures of a Distributed tool called Zookeeper, from installation, configuration and operation, the same ones that are made for the purpose of administering, support for large amounts of information, the selection of an application that resulted from the requirements of the distributed research that has database NoSQL, Also an analysis of Apache Zookeeper was realized where the service that offered Zookeeper like: Consensus, Choice of the leader and Creation / elimination of nodes. However, it was embodied in order to manage processes using both the Zookeeper and Cassandra service. And finally with the DataDog tool some metrics could be monitored as: performance of the equipment and operation of the Distributed Zookeeper and Cassandra system with the purpose of recommending the system suitable for use in distributed environments.

KEYWORDS: NoSQL, distributed systems, distributed processes, consensus.

INTRODUCCIÓN

La computación Distribuida es un área de interés, por lo que dos o más maquinas colaboran en la obtención de un resultado y están basados en características como eficiencia, flexibilidad, escalabilidad y fiabilidad. Cuyo objetivo principal de los sistemas Distribuidos es el mejor desempeño de fiabilidad, disponibilidad y compartición de recursos e información.

Hay algunas aplicaciones que permiten trabajar con computación Distribuida entre ellas están la propuesta por Google, denominada Google Cloud y la propuesta por Apache denominada Zookeeper, entre otras.

En este trabajo de titulación se propuesto analizar el funcionamiento de sistemas Distribuidos con la herramienta de Zookeeper y entender cómo se producen los diferentes procesos de control de fallos y consenso, las mismas que serán realizadas en estudios y proyectos posteriores.

Una vez analizada la herramienta de Zookeeper y Cassandra, se medirá el consumo de recursos que utiliza estos sistemas Distribuidos, con una herramienta estadística como es DataDog.

El presente trabajo denominado **“Análisis y funcionamiento de Zookeeper”**, para el entendimiento del mismo, se lo ha hecho en los siguientes 5 secciones:

CAPÍTULO I denominado **“Visionamiento”,** se identifica la problemática que ahora tienen los sistemas Distribuidos y como interviene Zookeeper para dar solución y poder implementarlo, después se da una breve introducción sobre los sistemas distribuidos para conocer sus inicios y propósitos.

CAPÍTULO II denominado “**Marco teórico“,** en la presente investigación de los sistemas Distribuidos se da a conocer: conceptos científicos, funcionamiento, aplicación, entre otros. Se desarrolló una indagación de la aplicación a implementar en un entorno Distribuido.

CAPÍTULO III corresponde **“Análisis”**, de los sistemas Distribuidos como son Zookeeper y Cassandra, primero se propone una indagación previa de cada uno de los entornos, luego se define la arquitectura, que está compuesta por estos procesos Distribuidos, se especifica los dispositivos donde se instala dichas aplicaciones y las comunicaciones que

se refieren a la conexión entre equipos para lograr la transmisión de los datos, por último se presentan pruebas obtenidas en el desarrollo para verificar si el funcionamiento obtenido cumple con el objetivo principal del proyecto.

CAPÍTULO IV denominado **“Análisis comparativo”,** se realiza la comparación de los sistemas Distribuidos Zookeeper y Cassandra, mediante la herramienta Datadog, que me permite calcular métricas de rendimiento del computador**,** donde se analizó resultados para ambos sistemas Distribuidos.

CAPÍTULO V denominado **“Conclusiones y Recomendaciones”,** sin embargo, finalmente la última sección corresponde el desarrollo de conclusiones y recomendaciones a partir de los objetivos del trabajo, también se expone trabajos que pueden desarrollarse a futuro que sean un aporte de desarrollo de sistemas Distribuidos.

CAPÍTULO I

VISIONAMIENTO

1.1. **Tema.**

Análisis de funcionalidad de Zookeeper

1.2. **Problemática.**

La computación Distribuida es un área de interés actual, hoy en día existen servicios tales como el almacenamiento en la nube, la edición síncrona de documentos. Sin embargo, hay algunas interrogantes que deben ser consideradas por ejemplo: a) Como trabaja el sistema en la nube, para consensuar un valor de almacenamiento de tal forma que el archivo siempre sea el más actualizado, b) En la edición síncrona en línea la coordinación de información de los datos cuando se almacena, c) Existe un servicio de computación Distribuido brindado a través de una granja de servidores, ¿cómo el sistema detecta que un servidor ha dejado de funcionar o en otro caso cuando un nuevo servidor se integra a la granja?.

Hay algunas aplicaciones que permiten trabajar con computación Distribuida entre ellas están la propuesta por Google, denominada Google Cloud y la propuesta por Apache denominada Zookeeper, entre otras. El presente trabajo de fin de titulación estudiará las características y funcionalidad de la herramienta Zookeeper. Se lo hará en algunas fases, desde la instalación de los sistemas, la integración con una aplicación y las pruebas de funcionalidad respectivas, con las nuevas y prometedores tecnologías presentes y que son futuro de la comunicación en el Internet.

1.3. **Alcance.**

El proyecto se centra en hacer un análisis de funcionamiento de Zookeeper, para entender cómo se resuelven y ejecutan los procesos de detección de fallos y consenso entre los dispositivos. Escoger una aplicación que soporte ambientes Distribuidos para el almacenamiento de información e implementación. Luego instalar un servicio que use las funcionalidades de los dos sistemas Distribuidos, y como estos interactúan entre ellos representándolos en datos estadísticos.

1.4. **Objetivos.**

1.4.1. Generales.

Estudiar y analizar la funcionalidad de una herramienta de computación Distribuida.

1.4.2. Específicos.

- Utilizar la herramienta de Zookeeper para determinar la funcionalidad de la computación Distribuida.
- Aplicar la computación Distribuida a una aplicación práctica.
- Describir el marco teórico de computación Distribuida y Zookeeper.

.

1.5. **Beneficios.**

Hoy en día los beneficios de los sistemas Distribuidos son varios y pueden ejecutarse en múltiples sistemas mediante la coordinación entre ellos, para completar una tarea en particular de una manera rápida y eficiente. A continuación mostramos beneficios.

Fiabilidad: El fallo de pocos o sólo un sistema no hace que todo el sistema falle.

Escalabilidad: El rendimiento puede incrementarse a medida que se necesita mediante la adición de más máquinas. Con menos cambios en la configuración de la aplicación sin tiempo de inactividad.

Transparencia: Oculta la complejidad del sistema y se muestra como una entidad única de aplicación.

1.6. **Terminología básica.**

En este punto indicamos conceptos básicos relacionados al trabajo de titulación a desarrollar, y así obtener una mejor capacitación y compresión de los temas planteados.

Nodo: Es el lugar donde se almacenan los datos.

Centro de datos: Se trata de una colección de nodos relacionados.

Clúster: Un clúster es un componente que contiene uno o más centros de datos.

NoSQL: Expresión asociada a la base de datos, son sistemas no relacionales.

Escalabilidad: Capacidad que tienen las bases de datos para mantener su calidad y funcionamiento, a pesar del crecimiento de los datos.

Ambiente Distribuido: Está conformado por un conjunto de equipos hardware físicamente conectadas entre sí, que transfieren información de un lugar a otro.

Detector de Fallos: En una herramienta Distribuida, cada proceso puede invocar para obtener información acerca de los fallos de los procesos. Hay muchas clases de detectores de fallos en función de la calidad y el tipo de la información devuelta.

Elección de Líder: Consiste en elegir un proceso entre varios, que se ejecutan en diferentes máquinas de un sistema Distribuido, para que actúe a modo de coordinador de **los** procesos.

Api: La interfaz de programación de aplicaciones (API), es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas, además usan funciones existentes en otro software.

CAPÍTULO II

MARCO TEÓRICO

2.1. Computación Distribuida.

En la actualidad la evaluación de los sistemas Distribuidos es de interés por la comunidad universitaria en general. La computación Distribuida surge y es básicamente un sistema de red el cual busca solucionar problemas de procesamiento de datos por medio de un conjunto de computadores o nodos. En estos casos existen algunos que son ligeramente acoplados se consideran independientes es decir, no admiten que se compartan memoria ni espacio de ejecución de los programas mientras hay otros sistemas de computación que son fuertemente acoplados comparten datos a través de un espacio de memoria común (M.L.Liu, 2004).

2.2. Términos y Conceptos.

A continuación tenemos algunas definiciones:

(Tanenbaum & Van Steen, 2007) afirma. “A distributed system is a collection of independent computers that appears to its users as a single coherent system” (p. 2).

(Coulouris, Dollimore, & Kindberg, 2012) define. “A distributed system is a set of computers which communicate network to coordinate their actions, as the sent of message” (pag.1).

(Lamport, 1978) también define. “Un sistema Distribuido consiste en un conjunto de procesos distintos que están espacialmente separados, y que se comunican mediante el intercambio de mensajes” (p.558).

Por lo tanto en este trabajo de titulación, un sistema Distribuido se basa en proporcionar al usuario y a las aplicaciones una visión de los recursos del sistema, gestionados por máquinas virtuales. Que a su vez permiten una distribución física transparente de los recursos.

2.3. Características de sistema Distribuidas.

Algunas de las características importantes de los sistemas distribuidos:

1. Son un conjunto de ordenadores independientes que almacenan un subred de comunicación donde permite la comunicación de los mensajes ya sea tanto hardware como software.

2. (Tanenbaum & Van Steen, 2007) nos dicen.” Otra característica importante es que los usuarios y aplicaciones pueden interactuar con un sistema distribuido de manera uniforme y consistente, independientemente de dónde y cuándo ocurre interacción” (p. 2).

3. (Coulouris et al., 2012) nos hablan de algunas características significativas. “Coincidencia de componentes, carencia de un reloj global y fracasos independientes de componentes” (p. 1).

   Sin embargo, los sistemas Distribuidos están compuestos por varios computadores conectados a una red de comunicaciones y equipados con programas que permiten coordinar sus actividades y compartir recursos.

2.4. **Requisitos de los sistemas Distribuidos.**

Los computadores y el acceso a red son económicos: Las computadoras de hoy en día tienen más potencia que las de antes además son de menor tamaño y precio, no obstante las computadoras están conectadas a causa de un componente que es computación Distribuida (M.L.Liu, 2004, p.8).

Compartición de recursos: La computación Distribuida ha causado un gran efecto dentro de las organizaciones al poder acceder a sus recursos más efectivamente, asimismo la Web es una plataforma muy potente donde permite compartir recursos dentro de las organizaciones (M.L.Liu, 2004, p.8).

Escalabilidad: Los recursos que estén disponibles estarán limitados de acuerdo con la capacidad de la computadora, por lo contrario la computación Distribuida proporciona escalabilidad que permite incrementar el número de recursos compartidos según la demanda (M.L.Liu, 2004, p.8).

Tolerancia a Fallos: La computación Distribuida permite que un recurso pueda ser replicado o reflejado, de tal forma que proporcione disponibilidad de los recursos en presencia de fallos (M.L.Liu, 2004, p.9).

Múltiples puntos de Fallos: La presencia de múltiples ordenadores que están implicados en la computación Distribuida todos son dependientes de la red para su comunicación, el fallo de uno o más computadores o enlaces de red puede suponer problemas para un sistema Distribuido.

Aspectos de seguridad: Los sistemas Distribuidos son vulnerables a fallos de seguridad y accesos no autorizados la cual pueden afectar a todos los participantes en el sistema, sin embargo estos problemas pueden ser por ataques bien conocidos en Internet como; los gusanos y virus. La seguridad juega un papel donde se hay técnicas de cifrado, claves, certificados, firmas digitales, entornos seguros, autenticación y autorización (M.L.Liu, 2004, p.9).

2.5. **Tipos de computación Distribuida.**

Para entender el funcionamiento y lo que significa la computación distribuida a continuación veremos diferentes formas de computación, según (M.L.Liu, 2004) nos afirma.

- **Computación monolítica:** Se especifica como un computador personal, estos no se encuentran conectado a ninguna red y se lo utiliza para aquellos procesos a los cual se puede acceder inmediatamente, además permite la convivencia de múltiples usuarios que comparten recursos de un único computador a través de una técnica de tiempo compartido.
- **Computación Distribuida***:* Es donde múltiples ordenadores se encuentran conectados a la red, cada uno tiene sus procesadores y otros recursos (mírese en la figura 1), por consiguiente el usuario utiliza su estación de trabajo y a la cual puede acceder al recurso del computador local conectado, asimismo a través de la interacción del ordenador local y computadores remotos se puede acceder a dichos recursos.

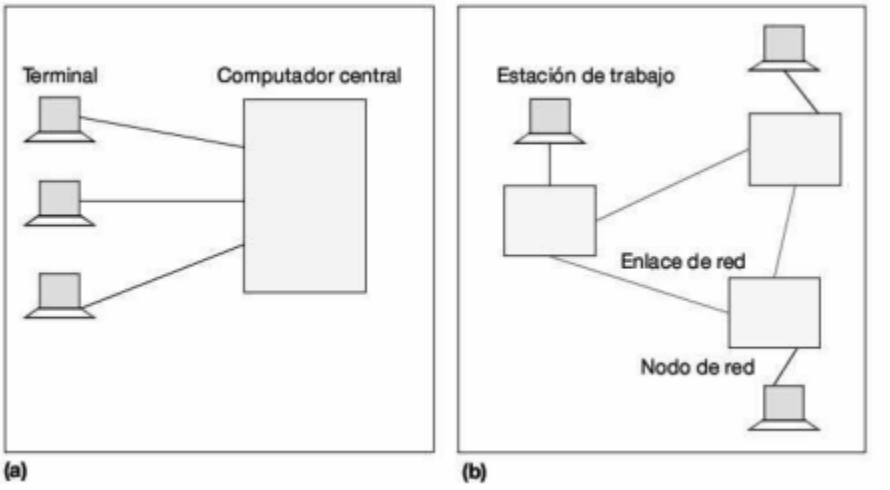


Figura 1: (a) Computación centralizada, (b) frente a computación Distribuida.
Fuente**:** (M.L.Liu, 2004).
Elaboración**:** (M.L.Liu, 2004).

- **Computación Paralela***:* Necesita más de un procesador simultáneamente para ejecutar un único programa, hoy en día se la utiliza en el área de biología, aeronáutica, etc. Aunque es una propuesta diferente a la computación Distribuida, se encuentra fuera de los objetivos de este proyecto.

- **Computación cooperativa*:*** Este tipo de computación divide a la computación a gran escala entre estaciones de trabajo de las máquinas del Internet, este sistema tampoco queda al alcance de este proyecto.

2.6. **Zookeeper.**

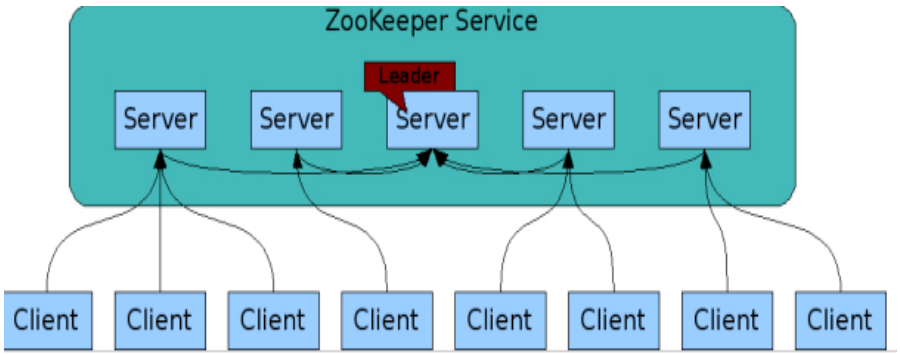


Figura 2: Modelo de servicio de Zookeeper.
Fuente: *"*The Apache Software Foundation (2013)*"*. **https://zookeeper.apache.org[1].**
Elaborado. The Apache Software Foundation

Zookeeper es un proyecto de Apache Software Foundation, provee un esfuerzo para desarrollar y mantener un servidor de código abierto que permite la coordinación distribuida altamente confiable.(The Apache Software Foundation, 2010). Sin embargo, fue diseñado para ser un servicio robusto lo cual permite a los desarrolladores centrarse en la aplicación de tareas de ordenación común como; gestión de la configuración, sincronización y servicios de grupo además permite elegir un servidor maestro, expone una Api simple.

Zookeeper almacena un conjunto de servidores así mismo permite tolerar fallas y escala de rendimiento, cuando se diseña una aplicación uno separa los datos de la aplicación (Soediono, 1989).

2.6.1. **Antecedentes de Investigación Zookeeper.**

Hoy en día existen grandes empresas que utilizan Zookeeper para aplicaciones industriales y así administrar procesos como: Yahoo, que utiliza Zookeeper para tareas de elección de líder, detección de fallos, almacenamiento de metadatos, y gestiona la recuperación de errores.

[1] Apache Software Foundation: https://zookeeper.apache.org (Consultado: 03/01/2017).

Facebook también maneja Zookeeper para: almacenamiento de registro de datos a través de múltiples máquinas, para conmutación de error y para detección de servicios (The Apache Software Foundation, 2013, p.9).

2.6.2. **Usos de Zookeeper.**

Zookeeper fue construido originalmente en “Yahoo” para acceder a sus aplicaciones de manera fácil y robusta, más tarde Zookeeper se convirtió en un estándar para los servicios apache, a continuación nombramos algunos mencionados por: (Soediono, 1989, p.5).

- **Apache HBase:** Es un almacén de datos que se utiliza normalmente junto con Hadoop, mientras que Zookeeper se utiliza para elegir a un maestro de grupo, además se hace un seguimiento a los servidores disponibles para así mantener los metadatos de clúster.

- **Apache Kafka:** Es un sistema de mensajería el cual Zookeeper permite detectar fallos.

- **Apache Solr**: Es una plataforma de búsqueda. En su forma Distribuida, llamado SolrCloud, utiliza a Zookeeper para almacenar metadatos sobre el grupo y coordinar las actualizaciones de estos metadatos.

- **Yahoo:** Este servicio utiliza a Zookeeper para tareas como la elección principal, detección y almacenamiento de metadatos.

2.6.3. **Ventajas y garantías de Zookeeper.**

ZooKeeper es una herramienta que permite hacer múltiples tareas de coordinación de una manera fácil gracias a su sencilla API, además representa una base para la construcción de los servicios más complicados, como por ejemplo la sincronización. También contiene muchas garantías de las cuales nos afirma (The Apache Software Foundation, 2013):

- **Consistencia Secuencial:** Se actualizará las peticiones del cliente y se aplicará el orden que fueron enviados.
- **Atomicidad:** Se realizan actualizaciones estos se presentan como éxito o fracaso.
- **Sistema único de imagen:** Un cliente verá el mismo punto de vista del servicio, independientemente del servidor que se conecta.
- **Confiabilidad:** Una vez que una actualización se ha aplicado, persistirá hasta el momento que un cliente sobrescribe la actualización.
- **Puntualidad:** La vista del sistema está garantizada para estar al día dentro de un plazo determinado (p.4).
- **Disponibilidad:** Zookeeper incrementa el rendimiento del procesamiento y garantiza la disponibilidad mientras haya mayoría de computadores activos y comunicados.
- **Durabilidad:** Los cambios persisten, a pesar de fallos.

2.6.4. **Servicios que ofrece Zookeeper.**

Zookeeper es un sistema Distribuido donde cuenta con servicios de coordinación de código abierto, sus procesos de alto rendimiento por lo cual lo pone a consentimiento la elección de líder, Estos servicios se utilizan para poder administrar los procesos de una forma ordenada, además permiten que se muestren notificaciones acerca de lo que sucede en los servidores, la motivación detrás de ZooKeeper es aliviar las aplicaciones distribuidas a cargo de la implementación de servicios de coordinación a partir de cero (The Apache Software Foundation, 2013):

a) **Elección del Líder:** Consiste en elegir un proceso de entre varios nodos, se ejecutan en diferentes máquinas, de un sistema Distribuido, para que hiciera a modo coordinador de los procesos.
b) **Consenso:** Permite a los procesos llegar a una decisión común a partir de algún valor inicial. Es decir, cuando se realiza una elección de líder, los computadores que funcionan como servidores se ponen de acuerdo y se establece el líder principal de un sistema Distribuido.

2.6.5. **Diseño de Zookeeper.**

Zookeeper es un servicio de coordinación Distribuido que, gracias a su diseño simple, permite centrarse en el funcionamiento de la aplicación, a continuación (The Apache Software Foundation, 2013, p. 2) define las siguientes características de Zookeeper:

- **ZooKeeper es simple:** ZooKeeper permite coordinar procesos entre sí, a través de un espacio de nombres jerárquicos que se organiza de manera similar a un sistema de archivos estándar. El espacio de nombre se compone de registros de datos llamadas znodes, y estos son similares a los archivos y directorios. A diferencia de un sistema de archivos que está diseñado para el almacenamiento de datos de ZooKeeper se mantiene en memoria, no obstante ZooKeeper puede lograr un alto rendimiento y bajos números de latencia.

  La implementación ZooKeeper es de alto rendimiento, alta disponibilidad de acceso, estrictamente ordenada, los aspectos de rendimiento de ZooKeeper significan que puede ser utilizado en sistemas grandes y Distribuidos.

- **ZooKeeper se replica:** Al igual que los procesos Distribuidos su coordinación, ZooKeeper se replica en múltiples hosts, llamados agrupaciones. Los servidores que componen el servicio ZooKeeper, Mantienen una imagen en memoria de estado, junto con los registros de transacciones en un almacén persistente. Mientras la mayoría de los servidores están disponibles, el servicio ZooKeeper estará disponible.

Los clientes se conectan a un único servidor ZooKeeper. El cliente mantiene una conexión TCP a través del cual se envía peticiones, consigue respuestas, adquiere ver eventos.

- **ZooKeeper se ordena:** Cada actualización con un número refleja el orden de todas las transacciones de Zookeeper. Las operaciones posteriores pueden utilizarlos con el fin de aplicar las abstracciones de nivel superior, como primitivas de sincronización.

- **ZooKeeper es rápido:** Es especialmente rápido en las cargas de trabajo. Zookeeper tiene aplicaciones que se ejecutan en miles de máquinas (The Apache Software Foundation, 2013, p. 3).

2.6.6. **Arquitectura de Zookeeper.**

La arquitectura es muy importante al momento de desarrollar un sistema ya que nos permite visualizar la estructura en la cual queremos producir un producto.

En la figura 3 se muestra la arquitectura de Zookeeper, los clientes se conectan al servidor donde hacen sus solicitudes de lectura son accesibles desde la réplica de su base de datos, las solicitudes que cambian el estado del servicio son las solicitudes de escritura.

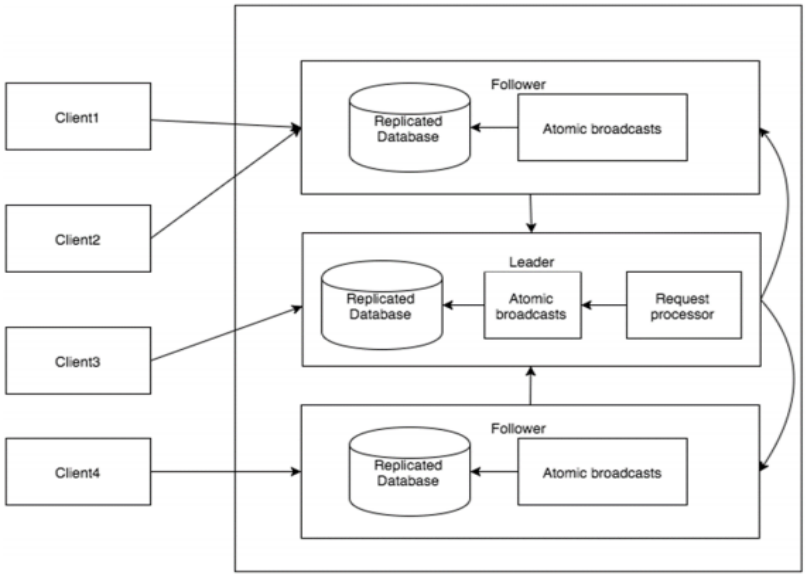


Figura 3: Arquitectura Zookeeper.
Fuente: "The Apache Software Foundation(2013)". ***https://zookeeper.apache.org²***
Elaborado:. *The Apache Software Foundation*

- La base de datos replicada: es una base de datos en memoria que contiene todo el árbol de datos. Las actualizaciones se inscriben en el disco de recuperación, y escrituras son serializadas en el disco antes de que se apliquen a la base de datos en memoria.
- Como parte del protocolo, todas las solicitudes de escritura de los clientes se envían a un servidor único. El resto de los servidores Zookeeper, llamados seguidores reciben los datos replicados por el líder. (The Apache Software Foundation, 2008ª, p. 6) .
- ZAB significa Zookeeper Atómic Broadcast este protocolo se utiliza para la replicación de transacciones, garantizando la consistencia, lo que significa que una vez que una transacción cumplió su objetivo, nunca se perderá siempre y cuando la mayoría de las máquinas estén trabajando correctamente.

Se identificaron dos tipos de mensajes relacionadas con Zookeeper:

- Los mensajes intercambiados entre los servidores (por ejemplo, durante la elección del líder).

- Los mensajes intercambiados entre un cliente y un servidor Zookeeper (por ejemplo, comando como crear, modificar, o borrar)

2.6.7. **Componentes del servicio Zookeeper.**

- **Emisión atómica**: Las solicitudes que actualizan el estado Zookeeper se reenvían al líder. El líder hace la solicitud y difunde el cambio en el estado ZooKeeper través de Zab. El servidor que recibe la petición del cliente responde al que haya efectuado el cambio de estado correspondiente.
- **Base de datos replicada:** Los datos de Zookeeper son replicados en múltiples nodos, por lo tanto, en caso de que un nodo caiga, los datos que se almacenan no se pierdan ya que otros nodos del clúster almacenan una copia de los mismos.

- **Procesador de solicitudes:** Al recibir una petición de escritura, el líder calcula en qué sistema estatal será una vez aplicada.
- **Interacción cliente-servidor:** Cuando un servidor trata una solicitud de escritura y envía que eliminen notificaciones relativas, cabe destacar que los servidores manejan notificaciones a nivel local.

2.6.8. **Modelo de datos Zookeeper**

La figura 4, presenta una estructura de árbol, donde Zookeeper maneja un sistema de archivos en memoria, cada nodo se conoce znode, cada znode se identifica por un nombre y separados por una ruta (/). (Zookeeper et al., 2015).

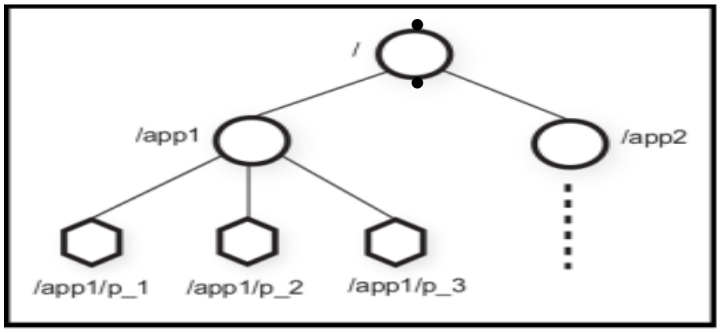


Figura 4: Espacio de nombre jerárquico.
Fuente: (Haloi, 2015)
Elaborado: Haloi

- En el diagrama, el nivel más alto contiene una raíz znode separado por (/). Bajo la misma raíz, tiene un espacio de nombres lógicos de configuración llamados trabajadores o hijos.
- Los znodes se llaman registro de datos, ya que se almacenan en la base de datos. Sin embargo, znode tiene hijos, así como los datos asociados a ella
- El espacio de nombres app se utiliza para una distribución centralizada, representado por el nivel medio.
- App contiene hijos znodes donde pueden almacenar 1MB de datos, como se puede ver en el nivel más bajo.

Cada znode en el modelo de datos ZooKeeper mantiene una estructura. Se compone de número de versión, lista de control de acción (ACL), la longitud de marca de hora, y datos (Zookeeper et al., 2015).

- **Número de versión:** El uso del número de versión es importante cuando varios clientes ZooKeeper están tratando de ejecutar operaciones en el mismo znode.
- **Lista de control de la acción (ACL):** ACL es básicamente un mecanismo de autenticación para acceder a la znode. Y regula todas las znode de lectura y escritura.
- **Marca de tiempo:** Representa el tiempo transcurrido desde la creación y modificación znode se representa en milisegundos. ZooKeeper identifica cada cambio en los znodes de "ID de transacción" (zxid). Zxid es único y que mantiene

la hora para cada transacción para que pueda identificar fácilmente el tiempo transcurrido de una petición con otra petición.

- **Longitud de datos:** Importa el total de los datos almacenados en una znode es la longitud de los datos. Puede almacenar un máximo de 1MB de datos.

2.6.9. **Tipos de Znodes.**

Zookeeper cuenta con tres tipos de znodes, persistentes, efímeros y secuenciales.

a) Znodes Persistentes

Los znodes persistentes tienen un tiempo de vida en el espacio de nombres de ZooKeeper hasta que son eliminados de forma explícita. Los znodes persistentes son útiles para el almacenamiento de datos son altamente disponibles y accesibles por todos los componentes de una aplicación distribuida. Por ejemplo: una aplicación puede almacenar los datos de configuración en un znode persistente.

A continuación (Haloi, 2015, p. 32) afirma:

- Un znode se puede eliminar llamando a la API de eliminación.
- No es necesario que sólo el cliente que ha creado un znode persistente deba eliminarlo
- Cualquier cliente autorizado del servicio ZooKeeper puede eliminar una znode.

b) Znode Efímero

Los znodes efímeros están activos mientras el cliente está vivo. Cuando un cliente se desconecta del conjunto ZooKeeper se eliminan automáticamente (Zookeeper et al., 2015) afirma: a) Los znodes efímeras no se les permite tener más hijos, b) Si se elimina un znode efímero, el siguiente nodo llenará su posición, c) Znodes efímeros juegan un papel importante en la elección del líder.

c) Znode Secuencial

Cuando se crea un nuevo znode como znode secuencial, ZooKeeper establece la ruta de la znode uniendo un número de secuencia de 10 dígitos al nombre original. Por ejemplo, si un znode con la ruta / “miaplicación” se crea como un znode secuencial, ZooKeeper va a cambiar la ruta de acceso / myapp0000000001 y establecer el siguiente número de secuencia como 0000000002. Si dos znodes secuenciales se crean al mismo tiempo, entonces ZooKeeper nunca usa el mismo número de cada znode. Los znodes secuenciales juegan un papel importante en el bloqueo y la sincronización (Zookeeper et al., 2015).

2.6.10. **Api de Zookeeper**

Según (Hunt, Konar, Junqueira, & Reed, 2014), Zookeeper consta de varios métodos para trabajar.

- **Create:** Crea nodos de Zookeeper.
- **Delete**: Elimina nodos que no se están utilizando.
- **Exists:** Verifica si el nodo existe para evitar nombres duplicados.
- **edata:** Obtiene información de los nodos de Zookeeper.
- **SetData:** Envía información de los nodos de Zookeeper.

2.6.11. **Consistencia de Zookeeper.**

ZooKeeper tiene operaciones que están diseñados para ser rápidas, aunque las solicitudes de lectura suelen ser más rápidas que escritura. Al mismo tiempo la lectura de ZooKeeper puede usar datos más antiguos, (Foundation, 2013, p.18).

Las solicitudes de lectura se procesan localmente en el servidor Zookeeper al cual el cliente está conectado, las peticiones de escritura se envían al líder y pasan por el consenso mayoritario antes de que se genere una respuesta.

- **La consistencia secuencial:** Las actualizaciones del cliente son destinados en el orden que fueron enviados.
- **Atomicidad:** Actualizaciones que tienen éxito o fracasan.
- **Sistema de una única imagen:** Un cliente verá el mismo punto de vista del servicio, independientemente del servidor que se conecta.
- **Confiabilidad:** Una vez que la actualización se ha aplicado, persistirá a partir de ese momento hasta que un cliente sobrescribe la actualización. Esta garantía tiene dos principios:

    1. Si un cliente recibe un código de retorno con éxito, se habrá aplicado la actualización. En algunos fallos (errores de comunicación, tiempos de espera, etc.) el cliente no sabrá si la actualización se ha aplicado o no. Se toma medidas para minimizar los fracasos, pero la garantía sólo se presenta con códigos de éxito.
    2. Zookeeper replica znodes en todo el clúster, por lo que si los servidores fallan, los datos de los znodes estarán disponibles, siempre y cuando la mayoría de servidores todavía estén funcionando.

2.6.12. **Herramienta para medir recursos Zookeeper.**

Uno de los objetivos de este proyecto de trabajo de titulación, tiene el fin de analizar Zookeeper y Cassandra, mediante una herramienta, que analice recursos de sistema, y asi definir mediante comparativa la mejor aplicación, gracias a la herramienta que medirá estos recursos, además se investigó otras aplicaciones, que desarrollan el mismo proceso para el análisis, de las cuales se detallan a continuación.

DataDog: Es un servicio de control de TI, operaciones y desarrollo de equipos que escriben y ejecutan aplicaciones en escala y quieren convertir las enormes cantidades de datos generados por sus aplicaciones,

Sematext: Es un sistema de control que va más allá de una recolección de métricas permite descubrir transacciones de base de datos lentos además permite mostrar su infraestructura en tiempo real, por otro lado es una herramienta de pago.

Grafana: Es una aplicación de código abierto que permite desarrollar cuadros estadístico, utilizando métricas en tiempo real, tanto para visualizar y analizar infraestructura de servidores.

2.6.13. **Herramienta para supervisión de Zookeeper y Cassandra.**

Se ha seleccionado la herramienta de DataDog, ya que es un servicio de TI, para equipos de desarrollo que escriben y ejecutan aplicaciones, convierten enormes cantidades de datos, generados por sus aplicaciones.

2.6.13.1. DataDog

DataDog es un servicio de vigilancia para aplicaciones que se ejecutan en la nube a gran escala, donde se reúnen datos de servidores, base de datos, herramientas y servicios para presentar una visión unificada de toda una pila. Datadog utiliza un agente de código abierto escrito en Python para recopilación de métricas y eventos, según (DataDog, n.d.).

DataDog supervisa infraestructura y rendimiento de las aplicaciones, se enfoca en el ítem siguiente:

- Detecta visión global y profundiza una solicitud de rastreo a una consulta SQL.
- Identifica cuellos de botella en el código de infraestructura.
- Relaciona el rendimiento de la aplicación o los eventos de métricas dentro de la infraestructura.

a) Funcionamiento de DataDog.

DataDog recoge métricas los analiza, procesa y supervisa los datos de rendimiento y luego gráfica. Es importante destacar que DataDog cuenta con un alto grado de eficiencia así como la amplia gama de servicios que permite integrar

b) Arquitectura DataDog.

Agente se compone de 4 componentes principales, cada uno escrito en Python se ejecuta como un proceso separado.

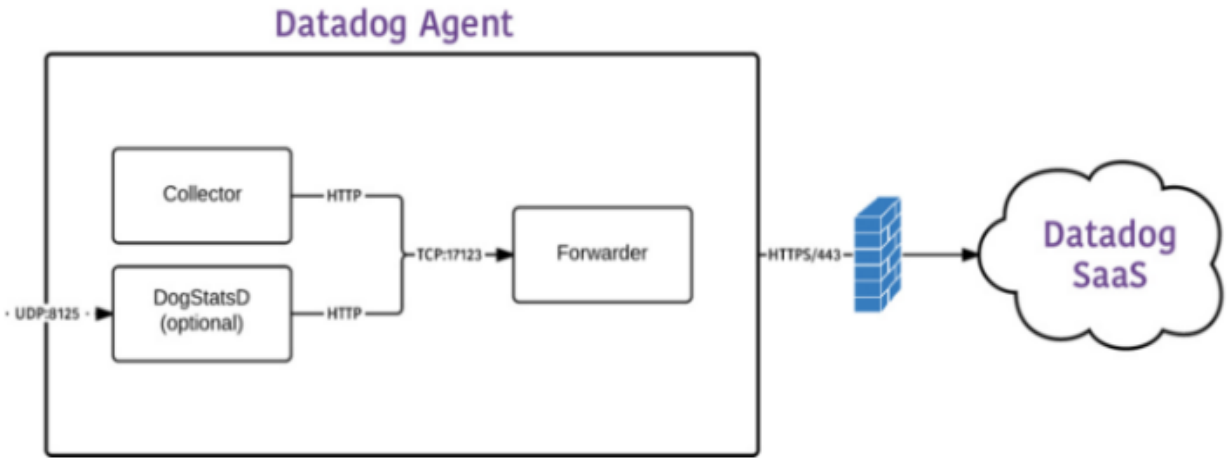


Figura 5: Arquitectura DataDog.
fuente**:** (Lawler, 2017)**.**
Elaboración**:** (Lawler)**.**

Permite escuchar a través de HTTP, para solicitudes entrantes luego a través de HTTPS a DATADOG. Permite a la red dividirse para no afectar presentación de métricas, las métricas se van almacenar en la memoria hasta que se alcanza un límite en el tamaño o número de solicitudes pendientes de enviar. Las métricas antiguas se descartaran de la memoria.

En La figura 5, se visualiza la arquitectura de DataDog, donde se especifica cada uno de los componentes del cual está estructurado y su cuál es su función internamente:

- **Collector (agent.py):** Se encarga de ejecutar controles de máquina actual por cualquier integración que se instale, el cual capta medidas del sistema como memoria y CPU, almacenados /etc/dd-agent/checks.d.

- **Dogstatsd (dogstatsd.py):** Este es un servidor back-end StatsD, que es responsable de la agregación de métricas locales**.**

- **Forwarder (ddagent.py):** Es un promotor de datos que trabaja junto a Dogstatsd y Collector y lo pone en cola para ser enviados a Datadog.

- **SupervisorD:** Todo es manejado por un proceso SupervisorD.

c) Presentación de datos de DataDog

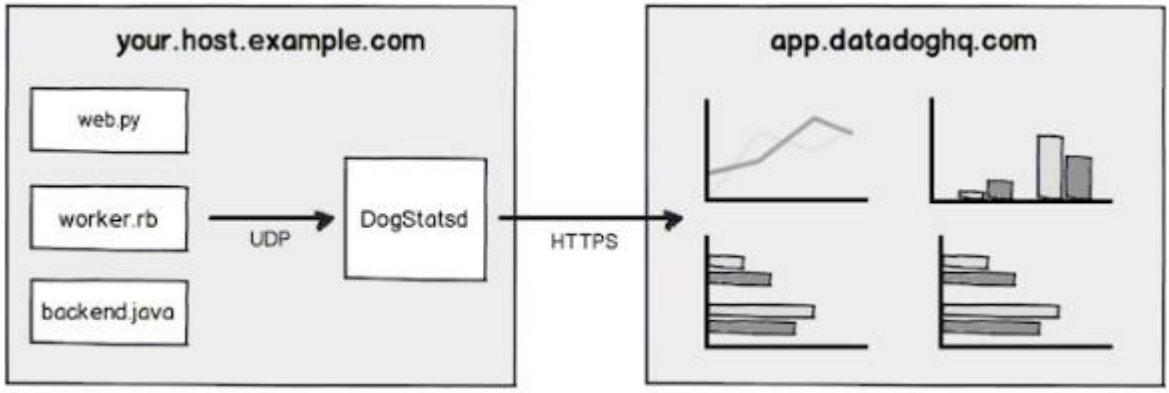


Figura 6: Modelo de datos de DataDog.
Fuente: (Fejoz, 2016).
Elaboración: (Fejoz, 2016).

En la figura 6, se visualiza el modelo de datos de DataDog, donde presenta el proceso de recolección de los datos, hasta su presentación gráfica, de los cuales son los siguientes:

- **Servidor Dogstatsd:** Esta integrado con el agente DataDog, encargado de recibir el valor muestreado, cada 10 segundos, se agrega los datos recibidos en los valores de indicadores.
- **Servicio Datadog:** Representa gráficamente los datos de las métricas.
- **Dogstatsd lado del cliente:** El cliente Dogstatsd envía datos con un incremento de 50% de veces.
- **Dogstatsd lado del Servidor:** Aquel que recibe el valor de contador.

2.6.13.2. StatsD.

Nos permite organizar aplicaciones utilizando bibliotecas de los clientes de acuerdo con un lenguaje específico, dichas bibliotecas se comunican luego con StatsD, utilizando un protocolo muy simple generando métricas agregadas para retransmitirlas a cualquier servidor backend de supervisión o creación de gráficos de acuerdo con (Olivier Pomel, n.d.).

1. Funcionamiento de los StatsD

El protocolo utilizado StatsD y servidor backend variará en función de servidor backend ya que la mayoría se fundamenta en HTTP. El servidor backend convertirá la corriente de datos en métricas de gráficos útiles y enviará notificaciones cuando sea necesario.

Todos los equipos de desarrollo y operaciones deben compartir la propiedad de rendimiento y disponibilidad de su aplicación.

2. Enviando métricas StatsD con Datadog.

Dentro de DataDog se ha empleado el protocolo StatsD, prestando mejoras para lo mismo, con el fin de añadir dimensiones a sus métricas, además cada host difundirá automáticamente sus métricas.

2.7. **Cassandra.**

Es una base de datos Distribuida que permite gestionar grandes cantidades de datos estructurados a través de muchos servidores posee un alto nivel de disponibilidad y ningún punto único de fallo.

Cassandra es una aplicación que usa un tipo de datos NoSQL, cuenta un sistema Distribuido similar a través de sus nodos, los datos se intercambian entre todos los nodos del clúster.

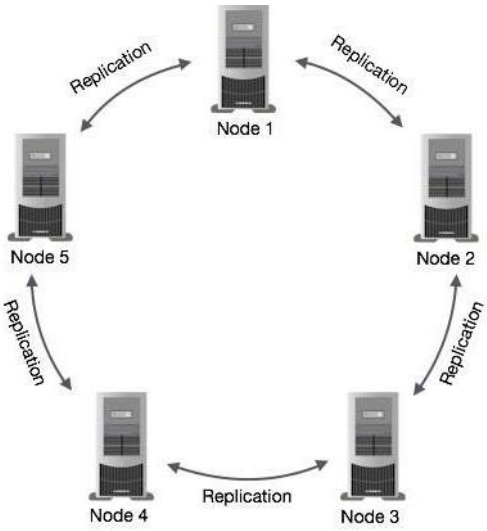


Figura 7: Modelo de Cassandra.
Fuente: (Guide, 2016).
Elaboración: Guide.

En la figura 7, Cassandra cuenta con un modelo de datos, que tiene un formato en anillo, donde la información se distribuye en varias nodos denominado como clúster, para llevar un control de fallos, donde cada nodo contiene una réplica de los mismos.
Según (Lee, 2017), lista características importantes sobre el modelado de datos.

- Todos los nodos de un clúster juegan el mismo rol, cada nodo es independiente y al mismo tiempo interconectado a otros nodos.
- Cada nodo de un clúster puede aceptar solicitudes de lectura y escritura, independientemente donde se encuentre dichos datos.
- Cuando un nodo se cae, las peticiones de lectura/escritura se pueden servir de otros nodos de la red.
- Si detecta Cassandra, que algunos de los nodos respondieron con valores diferentes de fecha, se lleva a cabo una actualización de aquel nodo.
- Sin embargo (Hewitt, 2011, p. 43) nos menciona: Cassandra se denomina con frecuencia como una base de datos orientados a columnas.

2.7.1. **Características de Cassandra.**

A continuación presentamos algunas características que lo han vuelto popular a Cassandra nos afirma (Guide, 2016).

Escalabilidad elástica: Cassandra permite añadir más hardware para dar desplazamiento a más clientes y datos según el requisito.

Arquitectura: Cassandra no tiene ningún punto único de fallo y es continuamente disponible para aplicaciones que no permiten un fracaso.

Rendimiento lineal de escala: Cassandra es escalable linealmente, es decir que aumenta su rendimiento a medida que aumenta el número de nodos del clúster. Por lo tanto, mantiene un tiempo de respuesta rápido.

Almacenamiento de datos flexibles: Cassandra contiene todos los formatos de datos, incluyendo: estructurada, semiestructurada, y no estructurados. Puede alojar de forma dinámica los cambios en sus estructuras de datos de acuerdo con su necesidad.

Distribución de datos: Proporciona flexibilidad para distribuir los datos que necesitan mediante la replicación de datos a través de múltiples centros de datos.

Soporte de transacciones: Cassandra admite propiedades como atomicidad, coherencia, aislamiento y durabilidad.

Escrituras rápidas: Cassandra fue diseñado para ejecutarse en hardware básico. Además lleva a cabo escrituras rápidas y almacenar cientos de terabytes de datos.

2.7.2. **Arquitectura Cassandra.**

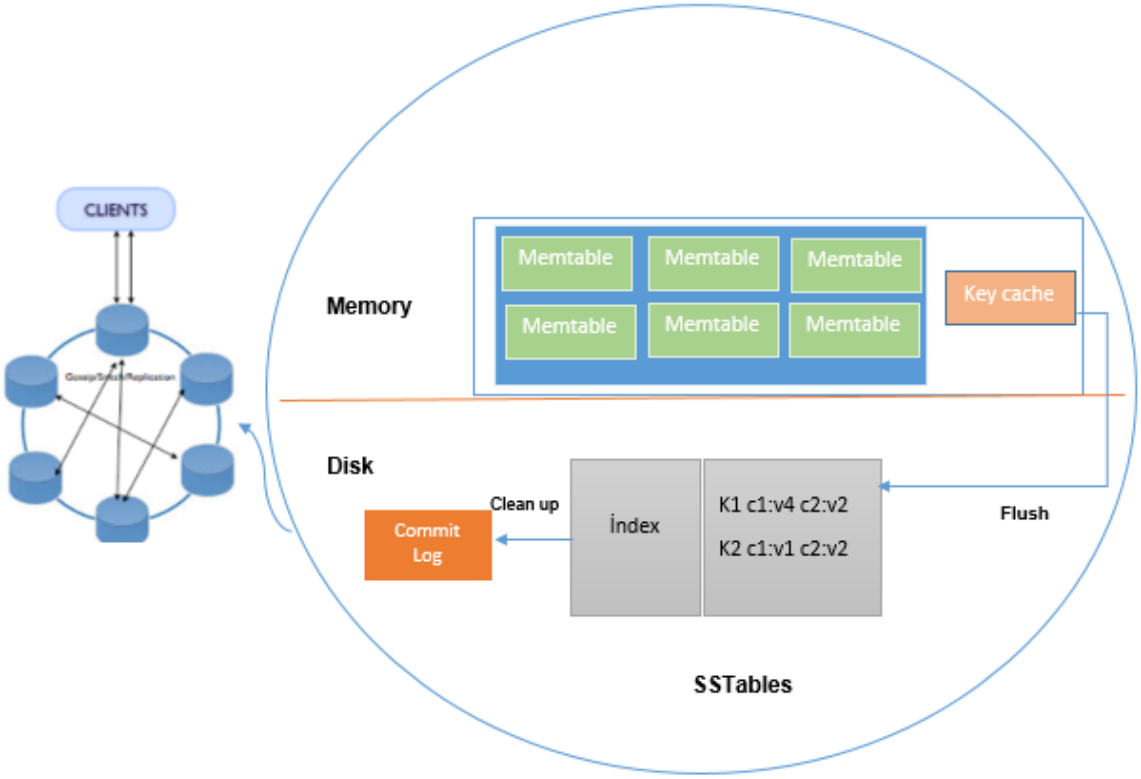


Figura 8: Arquitectura de Cassandra
Fuente: La Autora, Ximena Samaniego A.
Elaboración: La Autora, Ximena Samaniego A.

La figura 8, se visualiza la arquitectura de Cassandra. Donde el procesamiento de los datos, sigue una secuencia en cuanto a la escritura de datos, empezando por el registro de una escritura y terminando en compactación.

Según (Lee, 2017), Cassandra cuenta con componentes que conforma la arquitectura .

- Clúster: Es aquel que lo conforman varios nodos relacionados.
- Nodos: Lugar donde se almacenan los datos.
- Commit log: Es un mecanismo de recuperación de accidente en Cassandra, cada operación de escritura se escribe en el registro de confirmación.
- Memoria (Memtable): Es una estructura de datos residentes en memoria, luego de hacer una escritura de datos, los mismos se escriben en memoria

- SSTables: Es un archivo de disco, en que los datos se añaden desde memoria.
- Keycache: Contiene las claves dentro de la memoria distribuida en una tabla que contiene columnas, lo cual puede almacenar 200000 claves
- Limpieza (Flush): Cuando existen demasiadas escrituras, se realiza una limpieza de datos almacenados en tablas.

Sin embargo, Cassandra procesa los datos en varias etapas, empezando con el registro de escritura y su afirmación, su proceso es el siguiente (Lee, 2017).

- **El registro de datos, y su confirmación:** Cassandra almacena datos en una estructura en memoria, guarda en disco, la confirmación es realizada con éxito cuando se lee la escritura del nodo en Cassandra.
- **La escritura de datos en memoria:** Es donde se almacena todas las escrituras de Cassandra basándose en su clave, esta memoria escribe hasta llegar un límite y luego se elimina.
- **Limpieza o flushing de datos en la memoria:** Cuando hay demasiadas escrituras en disco, la velocidad de las transacciones disminuye, por lo que se debe hacer una limpieza a las tablas de Cassandra para mantener rendimiento.
- **Almacenamiento de datos en el disco SSTable:** es un archivo de disco, en que los datos se añaden a la memoria, pueden contener información obsoleta, por ejemplo: contiene un valor antiguo y un nuevo valor de la misma célula, o un valor antiguo para una célula nueva. Eso está muy bien, como Cassandra utiliza marcas de tiempo para cada valor o eliminación de averiguar cuál es el valor más reciente.
- **Compactación**: Cuando hay un flujo continuo de escrituras, esta acción lleva a una compactación eficiente.

Cassandra maneja grandes cargas de trabajo de datos a través de múltiples nodos sin ningún punto único de fallo, los datos se indexan y se escriben en memoria, asemejan a una cache de escritura no simultánea. Cada vez que esta memoria se llena, los datos se escriben en disco. Todas las escrituras se dividen y se replican en todo el clúster automáticamente. La arquitectura permite a cualquier usuario autorizado conectarse a cualquier nodo en diferentes centros de datos utilizando el lenguaje CQL.

A continuación se describen algunas de sus ventajas:

- Sus nodos tienen igual tratamiento.
- No hay jerarquía (no Maestro/Esclavo).
- Las filas se distribuyen mediante una función hash.
- Cada nodo es responsable de un rango de claves hash.
- La arquitectura puede ser vista como un anillo.
- Los datos se almacenan en tablas (familia de columnas).
- Las tablas se guardan en keyspace (base de datos).
- Las estructuras son flexibles.

2.7.3. **Estructura de Cassandra.**

En Cassandra, un espacio de claves o keyspace es el contenedor de datos de la aplicación, en el interior del espacio de claves existen uno o más grupos de familia de columnas, las mismas contienen columnas. De hecho, una buena forma de modelar la base de datos es diseñar una familia de columnas y así optimizar el rendimiento de lectura.

a) Keyspace.

Es un contenedor de datos, similar a un esquema de base de dato relacional, el keyspace se utiliza para agrupar familias de columnas.

Al definir un keyspace se configura siguientes parámetros (Guide, 2016):

- **Particionado:** Determina como se distribuyen los datos a través de los nodos del clúster.
- **Factor de replicación:** Establece número de nodos que actuarán como copias de un conjunto de datos, cantidad de veces que están repetidos los datos en el clúster.
- **Estrategia de ubicación:** Establece el modo de replicación de los datos en el clúster.

b) Columnas.

Es la estructura básica de datos de Cassandra, cuenta con tres valores los cuales son el nombre de la columna, valor y tiempo, a continuación en la figura 9, se muestra la estructura de una columna.

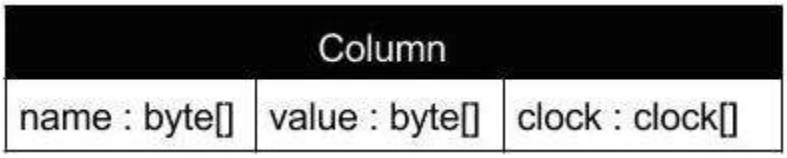


Figura 9: Estructura de columna.
Fuente:(Guide, 2016)
Elaboración:(Guide)

- **Nombre de columna (name):** Una columna debe tener un identificador, el cual puede ser una etiqueta estática, por ejemplo. Un nombre o mail, este valor debe ser único y no puede existir dos iguales en el conjunto de columnas.
- **Valor (value**): Corresponde al dato de la columna, es el único elemento modificable por el usuario, se puede definir la validación del tipo de dato que contiene.
- **Tiempo (clock):** Cassandra usa un tiempo de columna para determinar la actualización más reciente**,** cuando hay varias secciones de clientes que actualizan las columnas al mismo tiempo, la actualización más reciente es el que finalmente persiste.

c) Familia de columna.

Una familia de columna es similar a una tabla de modelo relacional, es un contenedor donde cada fila se encuentra ordenada, representa la estructura de los datos.

Keys_cache: Representa el número de ubicaciones para mantener en cache por SSTables.

Rows_cached: Representa el número de filas cuyo contenido entero se almacenan en cache.

Preload_row_cache: Se especifica si se desea rellenar previamente la memoria de cache de fila.

CAPÍTULO III

ANÁLISIS

En este capítulo III, se realizó la investigación de dos sistemas Distribuidos como son Zookeeper y Cassandra, los cuales usan una base de datos NoSQL, se considera pertinente comprender cada uno de estos sistemas Distribuidos, las mismas que permiten construir una comparativa general, para el desarrollo de trabajo de titulación y así analizar y elegir el sistema ideal.

3.1. **Comparación de sistemas Distribuidos Zookeeper y Cassandra.**

Tomando en cuenta la tabla 1, Se hace una comparativa donde se analizan los aspectos de funcionamiento de los sistemas Distribuidos según la naturaleza de la bases de datos, en los siguientes apartados se detallan los aspectos mencionados en la siguiente tabla.

Tabla 1: Comparación de sistemas Distribuidos.

Propiedades	Cassandra	Zookeeper
Propiedad	Apache	Apache
Tipo de almacenamiento	Columnas	Datos
Licencia	Libre	Libre
Sistemas Operativos	Windows, Ubuntu	Windows, Ubuntu
Alta disponibilidad	Alta, Cassandra permite replicación de datos, permite reflejar datos de otros nodos del clúster.	Alta, Zookeeper guarda sus datos en espacio de nombres jerárquico, los clientes pueden leer y escribir desde los nodos y de esta forma tiene un servicio de configuración compartido.
Tolerancia a fallos	Alta, configurados para la perdida de algunos nodos no interrumpen funcionamiento global.	Medio aún no se encuentran totalmente asegurados en cuanto a perdida de nodos.
Confiabilidad	Baja	Medio

Fuente: La Autora, Ximena Samaniego A

Elaboración: La Autora, Ximena Samaniego A

3.2. **Pruebas de rendimiento.**

Para las pruebas de rendimiento de las aplicaciones se utilizaron 3 computadoras, las características se muestran en la figura 10.

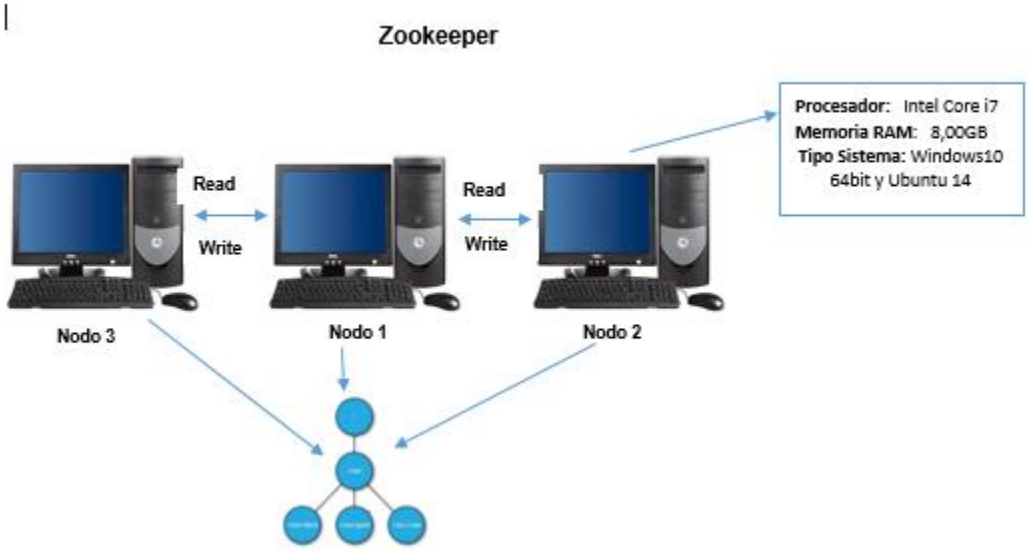


Figura 10: Pruebas de rendimiento Zookeeper
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A

En la figura 10, Se implementó Zookeeper tomando en cuenta las características de rendimiento y como se organizan jerárquicamente en forma de árbol, igual que un sistema de archivos.

Para iniciar con la instalación de Zookeeper primero se determinó el siguiente ítem.

- Instalar en 3 equipos los sistemas operativo Ubuntu con las mismas características 3 GB RAM, 500GB en disco.
- Montar Zookeeper en los 3 servidores.
- En los 3 equipos iniciar los servidores de Zookeeper uno a continuación de otro verificando funcionamiento de los mismos.
- Desarrollar el respectivo manual de instalación de Zookeeper, este procedimiento se encuentra en el anexo A.
- Desarrollar el manual de funcionamiento de Zookeeper, este procedimiento se encuentra en anexo E.

Para continuar con la instalación de Cassandra se considera lo siguiente:

- Instalación de Cassandra en 1 equipo con sistema operativo Ubuntu con las mismas características 3 GB RAM, 500GB en disco.
- Iniciar servidor Cassandra verificando funcionamiento del mismo.
- Desarrollar el respectivo manual de instalación de Cassandra, este procedimiento se encuentra en el anexo B.
- Desarrollar manual de funcionamiento de Cassandra, este proceso se encuentra en el anexo G.

Herramienta utilizada para el análisis de las aplicaciones de ZooKeeper y Cassandra.

- Instalación de herramienta gráfica DataDog para controlar procesos y servicios.
- Iniciar el monitoreo de los servidores mediante la aplicación DataDog.
- Mediante la herramienta gráfica instalada, determinar gráficamente consumo de recursos, de Zookeeper y Cassandra.
- Se desarrollará el respectivo análisis de los resultados de la aplicación gráfica de DataDog.

3.3. **Métricas.**

Para determinar la eficiencia computacional entre Cassandra y Zookeeper.

- Rendimiento: peticiones de lectura y escritura.
- Latencia: de lectura y escritura.
- Uso de disco: espacio de disco en cada nodo.

En el estudio presentado por (David Mytton, 2016) de acuerdo con la tabla 2 nos muestra algunas de las métricas más importantes acerca de Zookeeper y Cassandra, que me permitirá hacer el análisis donde se resaltan las principales características de este sistema, sin embargo, este proceso se encuentra en el anexo I.

Tabla 2: Métricas integradas

Métricas	Comentarios	Disponibilidad	
		Cassandra	Zookeeper
Solicitudes/ latencia Max	Cantidad de tiempo que toma para que el servidor responda a una petición del cliente (desde que se inició el servidor).	X	X
Recibidos	Número de solicitudes de cliente (típicamente operaciones) recibido.		X
Enviados	Número de paquetes enviados cliente (respuestas y notificaciones).		X
Archivos descriptor	Número de descriptores de archivos utilizados por encima del límite.		X
Numero de nodos	Número de znodes en el espacio de nombres Zookeeper		X
Uso de memoria	La memoria asignada dinámicamente por el proceso de Cassandra y Zookeeper	X	X
Uso CPU	Uso de de Cpu Utilizado con los procesos Cassandra Zookeeper	X	X
Uso memoria cache	Memoria cache asignado	X	X
Tamaño aproximado de datos	Numero de datos almacenados actualmente en los procesos		X

Fuente: La Autora, Ximena Samaniego A
Elaboración: La autora, Ximena Samaniego

Según (David Mytton, 2016; Matson, 2016), Cassandra y ZooKeeper ofrecen una gama de métricas en cuanto a rendimiento y utilización de recursos disponibles, se utilizó la herramienta de DataDog, el cual permitió usar todas las métricas antes mencionadas en la tabla 2.

- **Solicitudes de latencia Max:** La supervisión de latencia nos da una visión global sobre Cassandra y Zookeeper, a veces puede indicar problemas de desarrollo o cambio de patrones, lo que puede requerir ajustes en clúster, la latencia utiliza métricas las cuales puedan vigilar lectura y escritura en tiempo real.

- **Recibidos:** Número de solicitudes que el cliente recibe.

- **Enviados:** Número total de paquetes enviados.
- **Archivos descriptores:** Número de archivos descriptores utilizados por encima del límite
- **Numero de nodos:** Número de nodos creados.
- **Uso de memoria:** El uso de memoria en el sistema operativo Ubuntu cuando los servidores de Zookeeper y Cassandra están funcionando correctamente.
- **Uso memoria cache:** Memoria que permanece de manera temporal, para aquellos datos que son requeridos para hacer determinada función.
- **Uso de disco:** Espacio total en disco utilizado proporciona el número total de bytes del clúster, debe llegar al tamaño total del almacén de datos de Cassandra y Zookeeper.
- **Uso CPU:** Para la CPU hace referencia el porcentaje de utilización para los procesos de Cassandra y Zookeeper.
- **Tamaño aproximado de datos:** Total de datos almacenados, en tiempo real de los procesos.

a) Relación parcial de propiedades y métricas.

Tabla 3. Relación parcial de propiedades y métricas

Propiedades	Cassandra	Zookeeper	Métricas tomadas con Datadog
Propiedad	Apache	Apache	_
Tipo de almacenamiento	Columnas	Datos	_
Licencia	Libre	Libre	_
Sistemas Operativos	Windows, Ubuntu	Windows, Ubuntu	_
Alta disponibilidad	Alta, Cassandra permite replicación de datos, permite reflejar datos de otros nodos del clúster.	Alta, Zookeeper guarda sus datos en espacio de nombres jerárquico, los clientes pueden leer y escribir desde los nodos y de esta forma tiene un servicio de configuración compartido.	Paquetes recibidos (Zookeeper). Paquetes enviados (Zookeeper). Archivos Descriptor (Zookeeper). Numero de nodos conectados (Zookeeper). Solicitud latencia (Zookeeper, Cassandra). Uso memoria (Zookeeper, Cassandra). Uso CPU (Zookeeper, Cassandra). Uso memoria Cache (Zookeeper, Cassandra). Disco utilizado (Zookeeper, Cassandra).
Tolerancia a Fallos	Alta, configurados para la perdida de algunos nodos no interrumpen funcionamiento global.	Medio aún no se encuentran totalmente asegurados en cuanto a perdida de nodos.	Números de nodos efímeros (Zookeeper).
Confiabilidad	Baja	Medio	Solicitud latencia.

Fuente: La Autora, Ximena Samaniego A
Elaboración: La autora, Ximena Samaniego

3.4. **Análisis de DataDog de Zookeeper y Cassandra.**

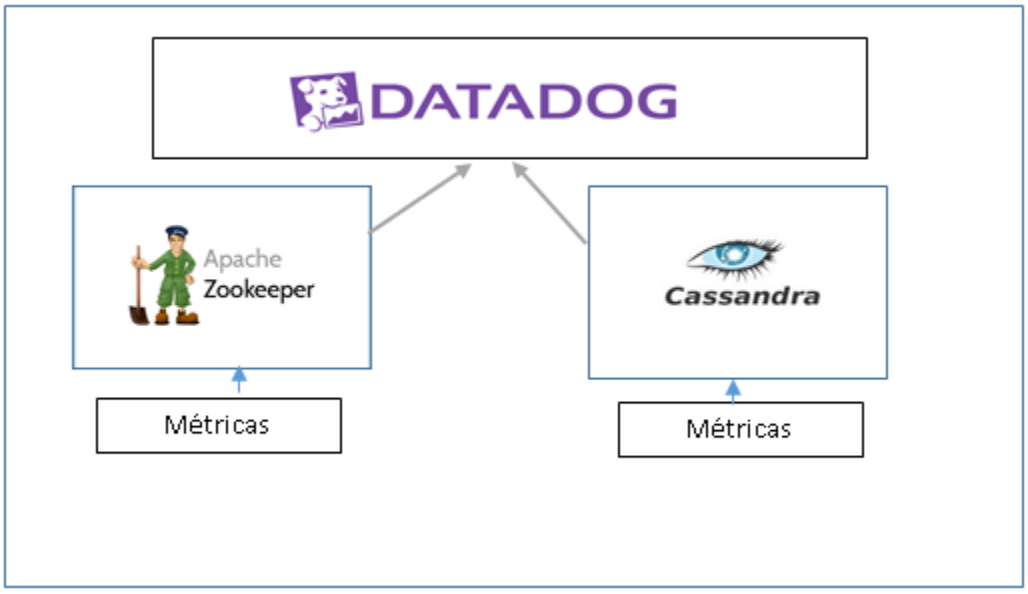


Figura 11: Arquitectura Zookeeper y Cassandra.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la figura 11, se da conocer cómo está estructurada nuestra plataforma a implementar con aplicaciones de Zookeeper y Cassandra en un entorno gráfico como DataDog.

DataDog nos permite controlar los hosts individuales, servicios, procesos y métricas, A continuación se desarrolla un estudio y análisis acerca de las métricas de Zookeeper y Cassandra.

Una vez que se ejecutó la aplicación de DataDog se procedió hacer la toma de datos del consumo de recursos del computador por ejemplo consumo de CPU, consumo de memoria, consumo de disco, consumo de red(Paquetes de entrada y paquetes de salida). Para lograr esto se utilizó la herramienta de DataDog para representación gráfica de datos ingresados.

3.4.1. **Conexión de Zookeeper.**

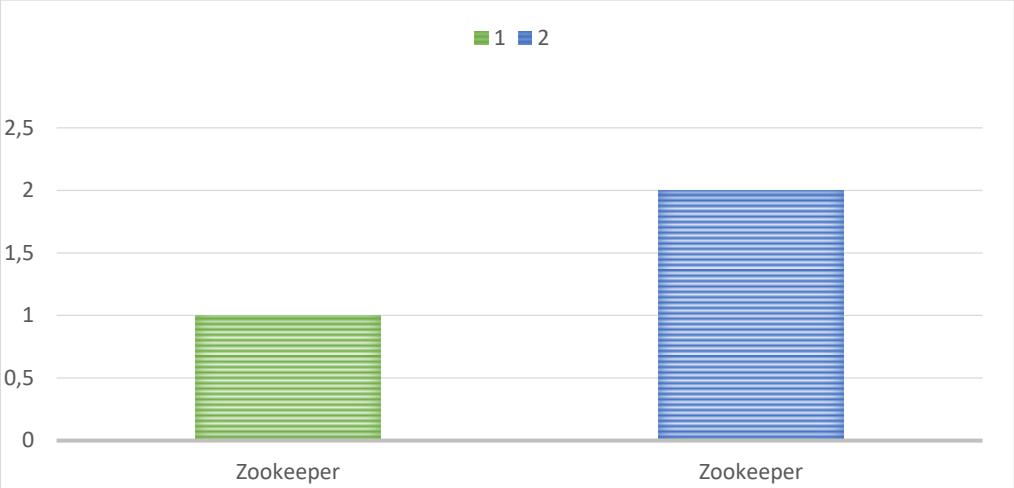


Figura 12: Conexión de clientes de Zookeeper.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

De acuerdo con los datos capturados en la figura 12, nos muestra la conexión exitosa de Zookeeper utilizando la métrica de **zookeeper.connections** muestra el número de conexiones que ha tenido con el cliente, es decir se ha realizado dos conexiones, por lo tanto cuando Zookeeper pasa más de un estado de conectado, ha tenido un funcionamiento normal.

3.4.2**. Tiempo de respuesta del cliente en Zookeeper.**

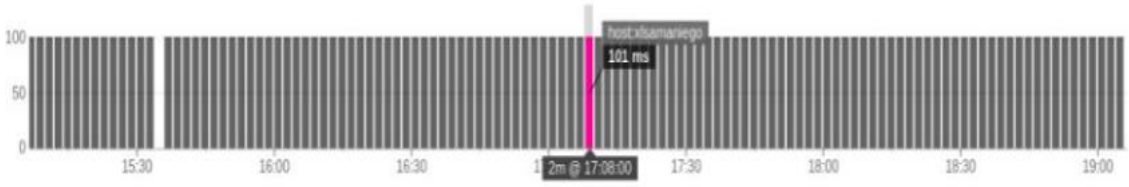


Figura 13: Tiempo de respuesta de solicitud del cliente.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Con la métrica zookeeper.latency.max se observa el tiempo que le toma al servidor a responder una solicitud del cliente y de acuerdo con la figura 13. Presenta un tiempo de 101ms que corresponde a milisegundos por lo cual este resultado se mantiene a lo largo del funcionamiento de Zookeeper.

3.4.3. **Número de paquetes enviados Zookeeper**

Aplicación	#número de paquetes enviados	Métrica utilizada
Apache Zookeeper	3	zookeeper.packets_sent

Figura 14: Paquetes enviados.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la figura 14, nos presenta el número total de paquetes enviados, utilizando la métrica Zookeeper.packets_sent el cual presenta un total de 3 paquetes enviados de Zookeeper.

3.4.4**. Número de paquetes recibidos Zookeeper.**

Aplicación	#número de paquetes recibidos	Métrica utilizada
Apache Zookeeper	3	zookeeper.packets_received

Figura 15: Paquetes recibidos.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Para el número de paquetes recibidos utilizamos métrica zookeeper.packets_received, en la figura 14 se presentó los paquetes enviados donde se muestra una similitud al estado de los paquetes recibidos de la figura 15, es decir que se ha enviado y recibido los paquetes con normal funcionamiento sin ninguna anomalía.

3.4.5. **Número de nodos creados**

Tabla 4: Numero de nodos conectados con ZooKeeper.

Aplicación	**#Nodos conectados actualmente**	**Métrica utilizada**
Apache Zookeeper	9	**(zookeeper.nodes),(zookeeper.znode_count)**

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Sin embargo, con la métrica **zookeeper.nodes** se visualiza el número de nodos creados con un total de 9 nodos.
También para visualizar el número de nodos se utiliza la métrica **zookeeper.znode_count** muestra un resultado idéntico al de tabla 4.

3.4.6. **Archivos disponibles para descriptores de Zookeeper.**

Figura 16: Descriptores Zookeeper.

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Tabla 5: Número de descriptores para Zookeeper.

Aplicación	#Máximo de descriptores	Resolución de descriptores	Métrica utilizada
Apache Zookeeper	99 datos	**4,07k**	**(zookeeper.max_file_descriptor), (zookeeper.open_file_descriptor)**

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A

A continuación con las métricas **zookeeper.max_file_descriptor** y **zookeeper.open_file_descriptor**, se puede visualizar el número máximo de archivos descriptores que un servidor puede abrir, en ZooKeeper cuyo resultado es de 4,07k de resolución, además permite un número de 99 archivos para descriptores ZooKeeper.

3.4.7. **Bytes de red enviados/recibidos pendientes por segundo.**

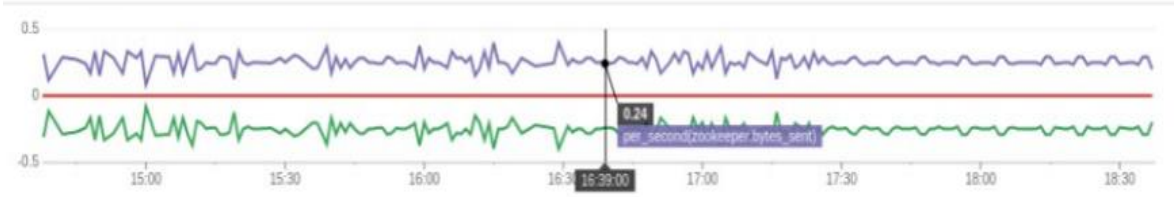


Figura 17: Descriptores Zookeeper.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Aquí obtenemos una imagen completa del uso de la red en el host, en el cual supervisa el rendimiento de la red a nivel del host de Zookeeper, lo cual observamos un rendimiento normal ya que tiene un nivel igual y su mayor frecuencia es hasta 1.

3.4.8. **Disco utilizado**

Tabla 6: Disco utilizado.

Aplicación	Disco utilizado
Apache Zookeeper	3.85 GB

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 6, Zookeeper hasta el momento utiliza **3.85** GB, de los 500 GB que posee el computador, que en términos generales utiliza 1% de disco.

3.4.9**. Uso de memoria cache**

Tabla 7: Uso de memoria cache.

Aplicación	RAM	Métrica utilizada	RAM(Byte)
Apache Zookeeper	844.49 MIB	System.mem_cached	88551.1 MB

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 7, visualiza el uso de memoria RAM física utilizada como memoria cache en

Zookeeper, utilizamos la métrica del sistema que es **System.mem_cached**, por lo que me da un total de 88551.1MB

3.4.10. Uso de memoria.

Tabla 8: Uso de memoria con Zookeeper.

Aplicación	Uso Memoria
Apache Zookeeper	3.34 GB

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 8, se observa el uso de memoria RAM que está ocupando Zookeeper, y se está usando es de 3,34 GB del total de memoria que es 8GB.

3.4.11. CPU utilizada

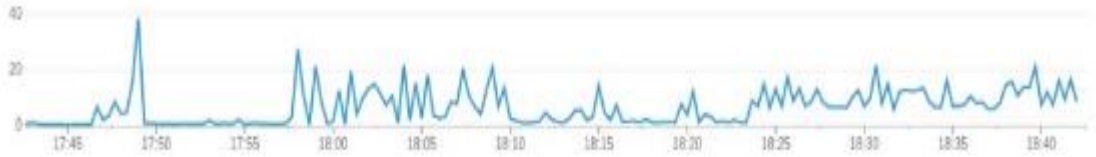


Figura 18: Uso de CPU con Zookeeper.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la figura 18, se visualiza un porcentaje de 39% de uso de CPU, donde se realizó la creación de nodos y lectura de los mismos, se nota una densidad alta el cual la CPU recorrió el Kernel, luego su procedimiento tiene un comportamiento normal.

3.4.12. Tamaño aproximado de datos en ZooKeeper

Tabla 9: Tamaño aproximado de datos en Zookeeper.

Aplicación	Tamaño de datos ZooKeeper	Métrica utilizada
Apache Zookeeper	164	zookeeper.approximate_data_size

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 9, nos muestra la media y máxima de la métrica del tamaño de datos de Zookeeper, es decir Zookeeper puede llegar almacenar 164 datos, se puede calcular dicho tamaño con la métrica siguiente **zookeeper.approximate_data_size**.

3.4.13. **Número de instancias utilizadas en Zookeeper**

Figura 19: Número de instancias en Zookeeper.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Dentro de la configuración de Zookeeper se ha podido determinar una instancia a lo largo del proceso tal como se especifica en la figura 19.

3.4.14. **Número de nodos efímeros creados en Zookeeper.**

Figura 20: Número de efímeros creados en Zookeeper.
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En el transcurso de creación de znode en Zookeeper se pudo crear efímeros. Estos znodes existen mientras la sesión que creó el znode está activo. Cuando termina la sesión se elimina el znode, esta gráfica nos visualiza un znode efímero creado.

a) Medición de datos con Cassandra en DataDog

Cassandra ofrece una gran variedad de métricas de rendimiento y utilización de recursos, dependiendo de las necesidades de cada métrica.

3.4.15. **Uso de memoria.**

Tabla 10: Uso de sistema de memoria

Aplicación	#Sistema de memoria	Métrica utilizada
Apache Cassandra	4.96GB	**System.mem.usable** **System.mem.total**

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En cuanto a Cassandra el uso de memoria RAM es de 3.33 GB, de lo que actualmente tiene la computadora que es de 8GB, donde ocupa casi un 40% de la memoria en sí.

3.4.16. **Uso de Cpu**

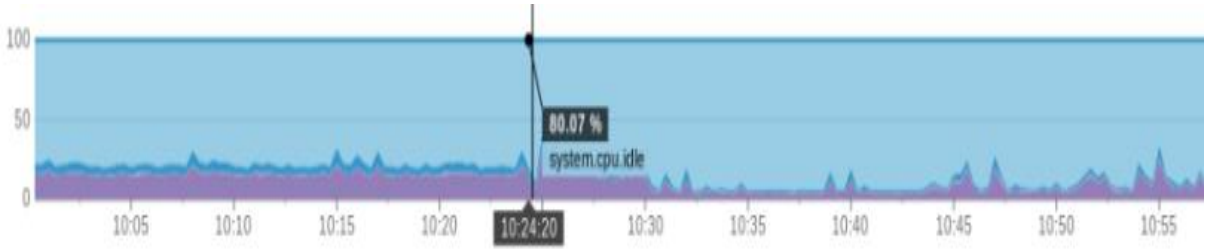


Figura 21: Uso de CPU
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Podemos observar la figura 21, que Cassandra utiliza un mayor porcentaje el uso de Cpu es 80% luego se nota una densidad normal del CPU.

3.4.17. **Uso de memoria cache**

Tabla 11: Uso de sistema de memoria

Aplicación	#Sistema de memoria cache	Métrica utilizada
Apache Cassandra	1.28GB	**System.mem.cache**

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 11, Cassandra en memoria cache utiliza 1,28GB, de acuerdo a su almacenamiento dentro del dispositivo.

3.4.18. **Escritura de latencia en Cassandra.**

Tabla 12: Escritura de Latencia en cassandra

Aplicación	#Escritura de Latencia	Métrica utilizada
Apache Cassandra	100ms	**Cassandra.db.recent_write_latency**

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

La escritura de latencia en Cassandra es de 100ms, el tiempo que le toma al servidor responder una solicitud del cliente un tiempo de 101ms que corresponde a milisegundos dicho resultados se mantienen a lo largo de la ejecución de la aplicación.

3.4.19. **Intervalo de actualización de Cassandra.**

Tabla 13: Intervalo de actualización de Cassandra

Aplicación	#Escritura de Latencia	Métrica utilizada
Apache Cassandra	100ms	**Cassandra.db.recent_write_latency**

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 13, La escritura de latencia e intervalo de actualización manejan los mismos tiempos ya que en Cassandra es de 100ms, el tiempo que le toma al servidor actualizar una solicitud del cliente.

CAPITULO IV

ANÁLISIS COMPARATIVO

Se desarrolla un análisis comparativo entre las métricas de rendimiento de Cassandra y Zookeeper, la recopilación de métricas de recursos nos pueden ayudar a reconstruir una imagen detallada del estado de nuestros sistemas, así tendremos una visión más detallada para futuras investigaciones y diagnóstico de problemas.

4.1. Recursos que consume DataDog.

DataDog utiliza un sistema basado en la nube permite recoger métricas a través de un sistema API del proveedor, del cual deben ser recuperados, por último tiene un proceso de visualización por pantalla, se tiene que dar importancia a los recursos que consume la herramienta de DataDog para su implementación.

Tabla 14: Recursos que consume DataDog

	Windows	Ubuntu
Memoria RAM	50 MB	50 MB
CPU	Menos 1% de tiempo de ejecución.	Menos 1% de tiempo de ejecución.
Disco	60 MB	120MB

Fuente: (Lawler, 2017)
Elaboración**:** (Lawler, 2017)**.**

Es importante destacar que DataDog en el trabajo de titulación, ha permitido comprender algunas dudas que se presentan dentro del proceso y así evitar problemas a futuro, con un diagnóstico en cuanto a soporte de servidores, redes físicas y recursos de almacenamiento.

DataDog utiliza un tiempo de ejecución de CPU menos del 1% promedio, en el Sistema Operativo Ubuntu necesita 120MB de Disco, mientras que en Windows 60MB, por el cual tiene una diferencia de la mitad procesamiento de CPU, el proyecto de titulación fue realizado en el sistema operativo Ubuntu ya que presta un proceso más ordenado en

cuanto a la estructura de archivos, se complementa muy bien a las aplicaciones de Zookeeper y Cassandra.

4.2. **Disco utilizado Cassandra y Zookeeper.**

Tabla 15: Disco utilizado de Cassandra y Zookeeper

Aplicación	Disco Utilizado	Métrica Utilizada	Descripción
Apache Zookeeper	3,85 GB	System.disk.total	Devuelve cantidad de disco utilizado en cuanto a ejecución y funcionamiento de creación, actualización de nodos de zookeeper
Apache Cassandra	9 GB	System disk.total	Devuelve cantidad de disco utilizado en cuanto a ejecución y funcionamiento, por creación de familias y columnas en cassandra.

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

Cabe destacar que la herramienta DataDog, me permitió comprobar el rendimiento de disco, en las aplicaciones de Cassandra como Zookeeper. Tomando en cuenta su funcionalidad con la creación, actualización y eliminación de nodos en Zookeeper, y Cassandra tiene creación, actualización, inserción de información en tablas NoSql.

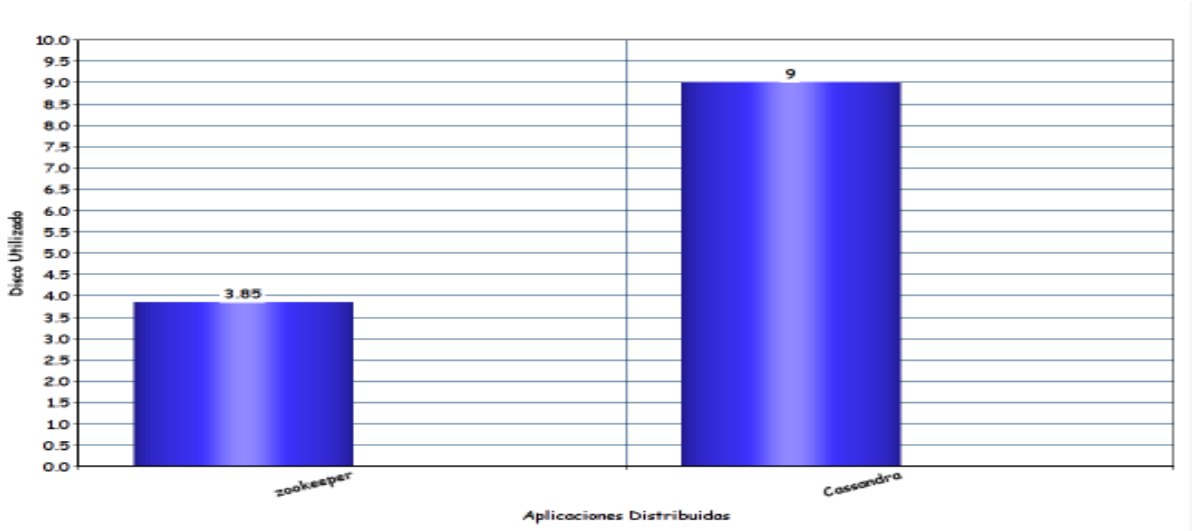


Figura 22: Disco utilizado de Cassandra y Zookeeper

Elaboración: La Autora, Ximena Samaniego A.

Sin embargo el número total de disco es de 500GB, tal y como consta en la figura 23 Zookeeper utilizo un total de 3,85 GB, que es casi 1% del equipo manejado, en cuanto a Cassandra tiene un consumo de disco de 9GB lo que significa un 2% del equipo utilizado,

4.3. **Uso de memoria cache.**

Tabla 16: Uso de memoria Cache

Aplicación	Memoria cache	Métrica utilizada	Descripción
Apache Zookeeper	88551,1MB	System.mem.cached	Devuelve cantidad de memoria cache utilizado en cuanto a ejecución y funcionamiento de creación, actualización de nodos de Zookeeper
Apache Cassandra	102701,7MB	System.mem.cached	Devuelve cantidad de memoria cache utilizado en cuanto a ejecución y funcionamiento, por creación de familias y columnas en cassandra.

Fuente: La Autora, Ximena Samaniego A

Elaboración: La Autora, Ximena Samaniego A.

La memoria cache cumple una función principal en cuanto almacenamiento de información del dispositivo, mejorando velocidad y eficiencia de Zookeeper y Cassandra, en la tabla 16, se demuestra el uso que tiene Zookeeper que es 844.49 MIB y Cassandra 979.44 MIB siendo este el que almacena mayor procesamiento de recursos.

4.4. **CPU utilizada.**

Se trabajó con un procesador total corei7 2.8GhZ para la ejecución de Cassandra y Zookeeper.

Tabla 17: CPU de utilizada de Cassandra y Zookeeper.

Aplicación	CPU	Métrica utilizada	Descripción
Apache Zookeeper	39%	System.cpu.user	Devuelve cantidad de CPU utilizado en cuanto a ejecución y funcionamiento de creación, actualización de nodos de zookeeper
Apache Cassandra	80%	System.cpu.user	Devuelve cantidad de CPU utilizado en cuanto a ejecución y funcionamiento, por creación de familias y columnas en cassandra.

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

De acuerdo con la tabla 17, se recopilo el tamaño utilizado en la CPU, por lo que Zookeeper trabaja con el 39% de procesamiento total, mientras Cassandra trabaja con el 80% del procesamiento total.

4.5. **Uso de memoria.**

Se Inició con una memoria de 8GB tota, para la ejecución de Cassandra y Zookeeper.

Tabla 18: Uso de memoria

Aplicación	Memoria	Métrica utilizada
Apache Zookeeper	3.34 Gb	System.mem.used
Apache Cassandra	4.96 Gb	System.mem.used

Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 18, se puede evidenciar claramente el incremento de consumo de recursos de Cassandra, en cambio Zookeeper consume menor cantidad de memoria.

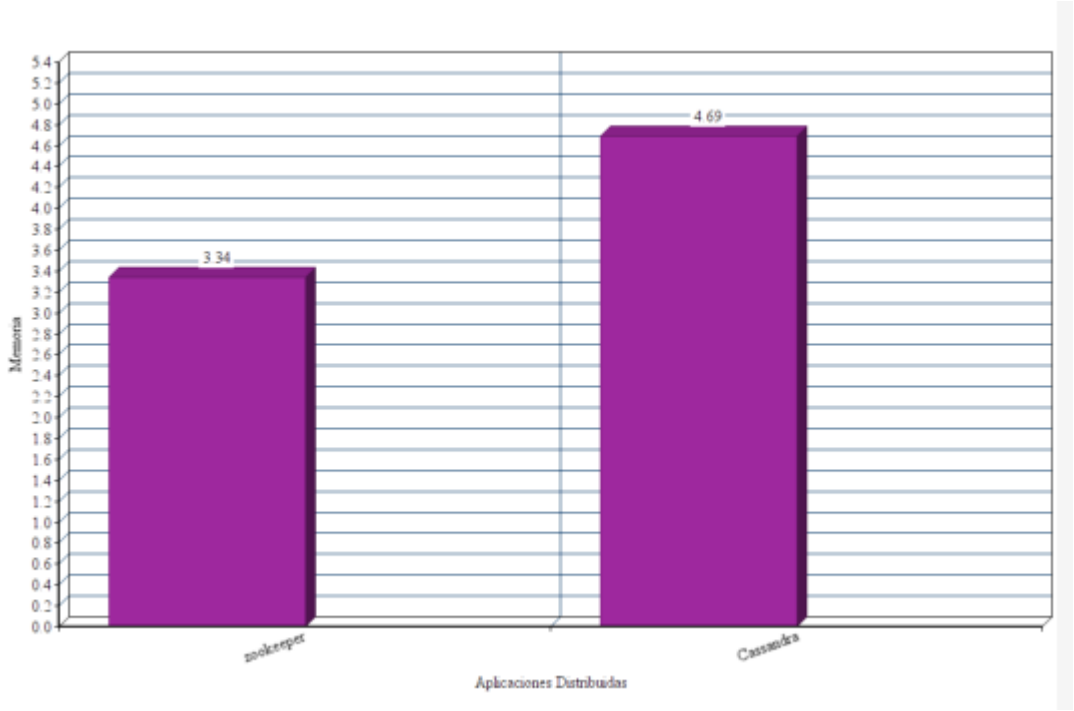


Figura 23: Uso memoria
Fuente: La Autora, Ximena Samaniego A
Elaboración: La Autora, Ximena Samaniego A.

En la tabla 18 y Figura 24 se puede concluir:

- ZooKeeper tiene un tamaño promedio 3.34 GB de uso de memoria cuando los servidores de Zookeeper están levantados.
- Cassandra tiene un promedio mayor de 4.96 GB, de uso de memoria cuando los servicios de Cassandra están corriendo.

GLOSARIO

StatsD: Es una instancia de red que se ejecuta en la plataforma Node.js y escucha estadísticas, como contadores y temporizadores enviados a través de UDP o TCP, envía a uno o más servicios de backend enchufables (grafito).

Latencia: Tipo especial que desarrolla un seguimiento de latencia (en microsegundos) con un Timer más una Counter que rastrea la latencia total acumulado desde que comenzó. El primero es útil si el seguimiento del cambio en la latencia total desde la última comprobación. Cada nombre de la métrica de este tipo tendrá 'latencia' y 'TotalLatency' adjunta a la misma.

Timer: Mide la tasa de una pieza particular de código más el histograma de su duración.

Counter: Es un indicador para una instancia, el cual es consumido por la monitorización desde la última llamada para saber si hay un gran aumento en comparación de la norma.

Nivel de consistencia: Define cuantos de los nodos replica deben responder a una solicitud de lectura y escritura.

SaaS: (Software as a Service) es un modelo de distribución de software donde el soporte lógico y los datos.

Disponibilidad: Mide los recursos del sistema que están disponibles para su uso para el usuario final a lo largo de tiempo.

Confiabilidad: Probabilidad de que un sistema cumpla con una misión específica.

CONCLUSIONES

Finalizado el proyecto de titulación se concluye lo siguiente:

Se han cumplido los objetivos planteados en el presente trabajo de fin de titulación, se instaló y se comprobó que Zookeeper es un servicio de coordinación de alto rendimiento que proporciona las herramientas necesarias para administrar aplicaciones distribuidas para el control de información, a través del clúster, sin preocuparse de perdida de información.

Del estado del arte se puede concluir que Zookeeper demuestra ser una herramienta centralizada, escalable y tolerante a fallos, permite la configuración del clúster con el propósito de almacenar varios nodos y sincronizar las tareas de los nodos y este sea tolerante a fallos. Zookeeper es compatible con muchas aplicaciones industriales. Zookeeper permite distribuir información al resto de nodos, por lo que si se genera un punto de fallo, el nodo siguiente asume el rol y recupera la información que se está transmitiendo. También se investigó más acerca del funcionamiento de Zookeeper y Cassandra basado en un sistema distribuido y sus ventajas. Cassandra ha integrado funcionalidades de Apache Hadoop y Apache Hive, para usar funciones y facilitar la extracción, transformación y carga de datos. Cassandra posee una base de datos NoSQL, este le permite almacenar y recuperar datos sin esquema alguno, además su replicación es fácil ya que maneja una API sencilla lo que le ayuda a manejar grandes cantidades de datos

Zookeeper tiene un mejor rendimiento que Cassandra, por lo que se puede concluir que la carga computacional de Zookeeper es menor a la de Cassandra.

Cassandra ocupa una mayor cantidad de recursos que Zookeeper.

En el capítulo III, se comprobó que Zookeeper es un servicio de coordinación estable, simple y de alto rendimiento que proporciona las herramientas necesarias para administrar aplicaciones distribuidas, con la simulación aplicada a Zookeeper alcanza valores de rendimiento y cientos de operaciones por segundo para cargas de trabajo.

En el capítulo IV, se comparó las aplicaciones de Zookeeper y Cassandra, utilizando la herramienta de Datadog que mediante a métricas se determinó el rendimiento computacional que ocupa cada uno de estas aplicaciones como:

- En cuanto a disco, se manejó un total de 500gb, Zookeeper en cuanto a ejecución, funcionamiento, actualización y creación de nodos, ocupo un mínimo rendimiento del 1% del total del disco empleado mientras Cassandra en cuanto a ejecución y funcionamiento por creación de familias y columnas en Cassandra manejó un rendimiento mayor de 2% del total del disco.
- En procesamiento de CPU, se trabajó con un procesador COREi7 2.8GhZ, por lo que Zookeeper utilizó un mínimo de 39% del procesamiento total, mientras Cassandra uso un 80% es decir una diferencia 41 puntos más del de Zookeeper.
- Respecto a uso de memoria, se utilizó un total de 8GB, Zookeeper utilizo un 30% de memoria total, mientras Cassandra obtuvo un 50% empleado, es decir un 20% más al de Zookeeper.
- Con DataDog, se integró algunas métricas adicionales para Zookeeper, en cuanto a total de nodos conectados, latencia, estado del host, tiempo de respuesta, numero de paquetes enviados y recibidos, tipo de archivos descriptores y a que resolución se establecen, lo que es ventajoso para un servidor como Zookeeper ya que me presenta un visión más profunda acerca del funcionamiento del mismo.

RECOMENDACIONES

Para la instalación de Zookeeper en computadora, es necesario revisar los requerimientos de hardware o software que nos da la aplicación para su respectiva instalación y asimismo evitar errores de hardware.

Cuando se está instalando las aplicaciones de Zookeeper y Cassandra es necesario establecer variables de entorno dentro del archivo .bashrc en el sistema operativo Ubuntu.

Es necesario contar con versiones actualizadas tanto de Python y Jdk8 que servirán para el correcto funcionamiento de Zookeeper y Cassandra.

Cabe recalcar que DataDog es una herramienta de pago, por lo que nos ofrece un paquete gratis de prueba de 15 días para monitorear máximo dos integraciones.

Para la toma del rendimiento de recursos del computador con DataDog, es necesario tener abierto solo la aplicación de Zookeeper o Cassandra, por lo que es recomendable cerrar otras aplicaciones que podrían utilizar los recursos del computador, de esta forma evidenciar el consumo que se genera Zookeeper y Cassandra.

BIBLIOGRAFÍA

Copyright. (n.d.). Tutorial de Zookeeper. Retrieved from https://www.tutorialspoint.com/zookeeper/zookeeper_applications.htm. 2016.02.06.

Coulouris, G., Dollimore, J., & Kindberg, T. (2012). *Distributed Systems: Concepts and Design*. *Computer* (Vol. 4). Retrieved from http://www.amazon.com/dp/0321263545. 2016.02.06.

DataDog. (n.d.). DataDog General. Retrieved from http://docs.datadoghq.com/. 2016.02.07.

David Mytton. (2016). How to monitor zookeeper. Retrieved from https://blog.serverdensity.com/how-to-monitor-zookeeper/.2016.02.07.

Fejoz, M. (2016). Presentacion de datos con Datadog. Retrieved from https://help.datadoghq.com/hc/en-us/articles/208398693--dog-statsd-sample-rate-parameter-explained.2016.03.02

Foundation, A. S. (2013). ZooKeeper Programmer ' s Guide, 1–24. Retrieved from https://zookeeper.apache.org/. 2016.03.02.

Guide, C. Q. (2016). CASSANDRA - INTRODUCTION CASSANDRA - ARCHITECTURE, 60.

Haloi, S. (2015). *Apache ZooKeeper Essentials*.

Hewitt, E. (2011). *The Definitive Guide*.

Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. (2014). ZooKeeper : Wait-free coordination for Internet-scale systems.

John Matson. (2015). How to monitor Cassandra. Retrieved from https://www.datadoghq.com/blog/how-to-monitor-cassandra-performance-metrics/. 2016.06.03

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21*(7), 558–565. http://doi.org/10.1145/359545.359563. 2016.06.04.

Lawler, D. (2017). Arquitectura DataDog. Retrieved from https://help.datadoghq.com/hc/en-us/articles/203034929-What-is-the-Datadog-Agent-What-resources-does-it-consume-

Lee, J. (2017). Tutorial de Cassandra. Retrieved from http://www.w3ii.com/es/cassandra/cassandra_data_model.html.2017.01.03

M.L.Liu. (2004). *Computación Distribuida: Fundamentos y Aplicaciones*.

Matson, J. (2016). Principales metricas de rendimiento de Cassandra. Retrieved from https://www.datadoghq.com/blog/how-to-monitor-cassandra-performance-metrics/.2017.01.04

Olivier Pomel. (n.d.). StatsD. Retrieved from https://www.datadoghq.com/blog/statsd/.2017.02.05

Soediono, B. (1989). *Zookeeper*. *Journal of Chemical Information and Modeling* (Vol. 53). http://doi.org/10.1017/CBO9781107415324.004.2017.02.05

Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms, 2/E*. *Communication*. http://doi.org/10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C.2016.07.05

The Apache Software Foundation. (2008). A Distributed Coordination Service for Distributed Applications “Zookeeper,” 1–10. Retrieved from https://zookeeper.apache.org. 2015.12.05

The Apache Software Foundation. (2010). Apache Zookeeper. Retrieved from https://zookeeper.apache.org/.2015.12.03

The Apache Software Foundation. (2013). ZooKeeper, 1–9. Retrieved from https://zookeeper.apache.org/.2015.12.03

Zookeeper, T., Zookeeper, A., Hbase, A., Analytics, B. D., Point, T., Point, T., & Point, T. (2015). About the Tutorial Copyright & Disclaimer, 39. Retrieved from http://www.tutorialspoint.com/zookeeper/zookeeper_leader_election.htm.2016.01.03

ANEXOS

ANEXO A: Instalación de Zookeeper.

Para la instalación de Zookeeper es necesario tomar en cuenta los siguientes requerimientos para su correcto funcionamiento.

1. Requerimientos del Sistema.

- **GNU / Linux** (Es compatible como plataforma de desarrollo y producción para servidor y cliente).
- **Sun Solaris** (Es compatible como plataforma de desarrollo y producción para servidor y cliente).
- **FreeBSD** (Es compatible como plataforma de desarrollo y producción para servidor y cliente).
- **Win32** (Se admite como una plataforma desarrollo solo para el servidor y cliente).
- **Win64** (Se admite como una plataforma desarrollo solo para el servidor y cliente).
- **MacOSX** (Se admite como una plataforma desarrollo solo para el servidor y cliente).

2. Requerimientos Software.

Zookeeper se ejecuta en java sea JDK6 o superior, se ejecuta como un conjunto de servidores empleando Zookeeper, además tres servidores es el tamaño mínimo recomendado para un conjunto y también se recomienda que se ejecuten en máquinas separadas se debe tomar en cuenta que se debería trabajar, con procesadores de doble núcleo 2GB de RAM y disco duro de 80 GB IDE.

Es necesario contar con un número impar de máquinas Zookeeper que pueda manejar el fallo de una máquina. Generalmente tres servidores es más que suficiente para instalar una producción con tres servidores, se realiza el mantenimiento de uno de ellos, que son vulnerables a un fallo en los otros dos servidores. Por lo tanto he decidido instalarlo en tres nodos y uno de ellos será un nodo maestro.

3. **Instalación de Java8 en Ubuntu con JDK8.**

1. Se instalara Java (JRE/ JDK) por defecto ya que instalara el OpenJDK8 en Ubuntu 14.04. Instalando Java con apt-get es fácil.

   ```
   sudo apt-get update
   ```

2. Después revisamos si Java no se ha instalado previamente:

   ```
   java -version
   ```

3. Si ese comando regresa “ The program java can be found in the folowing packages”, significa que Java no está instalado por consiguiente procedemos a instalar con siguiente comando de acuerdo a la figura 1.

   ```
   sudo apt-get install default-jre
   ```

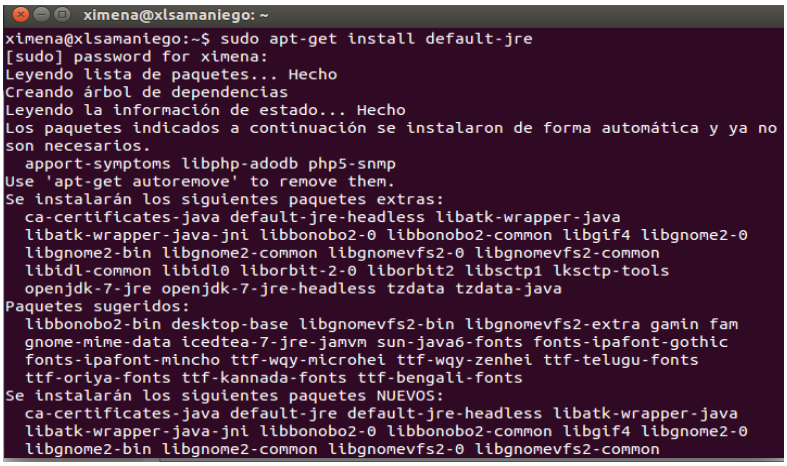


Figura 1. Comando para instalación de java
Elaboración: La Autora, Ximena Samaniego A.

4. Instalación de apache Zookeeper en Ubuntu.

Con el siguiente comando nos permitirá ir a la página de ZooKeeper y descargar la última versión estable de ZooKeeper en nuestro caso se instaló la versión de “Zookeeper-3.4.8” tal y como está ilustrado en la figura 2.

```
sudo wget
http://mirrors.ukfast.co.uk/sites/ftp.apache.org/zookeeper/stable/zookeeper-3.4.8.tar.gz
```

```
ximena@xlsamaniego:~$ sudo wget http://mirrors.ukfast.co.uk/sites/ftp.apache.org
/zookeeper/stable/zookeeper-3.4.8.tar.gz
[sudo] password for ximena:
--2016-03-07 17:21:29--  http://mirrors.ukfast.co.uk/sites/ftp.apache.org/zookee
per/stable/zookeeper-3.4.8.tar.gz
Resolviendo mirrors.ukfast.co.uk (mirrors.ukfast.co.uk)... 78.109.175.117
Conectando con mirrors.ukfast.co.uk (mirrors.ukfast.co.uk)[78.109.175.117]:80...
 conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 22261552 (21M) [application/x-gzip]
Grabando a: “zookeeper-3.4.8.tar.gz”

100%[======================================>] 22.261.552   581KB/s   en 3m 57s

2016-03-07 17:25:27 (91,6 KB/s) - “zookeeper-3.4.8.tar.gz” guardado [22261552/22
261552]
```

Figura 2: Comando para descargar la última versión estable de Zookeeper
Elaboración: La Autora, Ximena Samaniego A.

4. Luego extraemos el archivo que está dentro del tar.gz:

sudo tar -xvf zookeeper-3.4.8.tar.gz

5. Nos dirigimos a la carpeta de ZooKeeper y entramos a la carpeta de conf para poder modificar un archivo zoo.cfg:

cd zookeeper-3.4.8/
cd conf

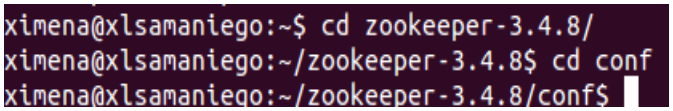


Figura 3: Comando para acceder a los archivos de Zookeeper y de configuración.
Elaboración**:** La Autora, Ximena Samaniego A.

6. Ya que estamos dentro del registro de configuración vamos a modificar el archivo de zoo.cfg con el siguiente comando ilustrado en la figura 4:

nano conf/zoo.cfg

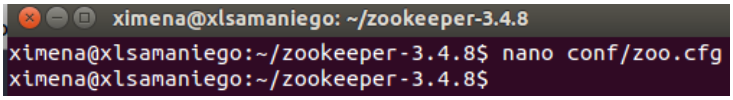


Figura 4: Comando para acceder a los archivos de configuración.
Elaboración: La Autora, Ximena Samaniego A.

7. Dentro del nano colocamos las siguientes líneas especificando los siguientes ítems:

La unidad de tiempo básica en milisegundos utilizado por ZooKeeper.
tickTime=2000

La ubicación para almacenar las instantáneas de base de datos en memoria.
dataDir=/var/lib/zookeeper

#Puerto de escucha para las conexiones de clientes.
lientPort=211

Tiempos de espera, utiliza para limitar la cantidad de tiempo en los servidores Zookeeper

initTime=5

Límites hasta qué punto fuera de fecha un servidor pueden ser de un líder.
syncTime=2

Luego de haber modificado dicho archivo procedemos a guardarlo con el mismo nombre ilustrado en la figura 5.

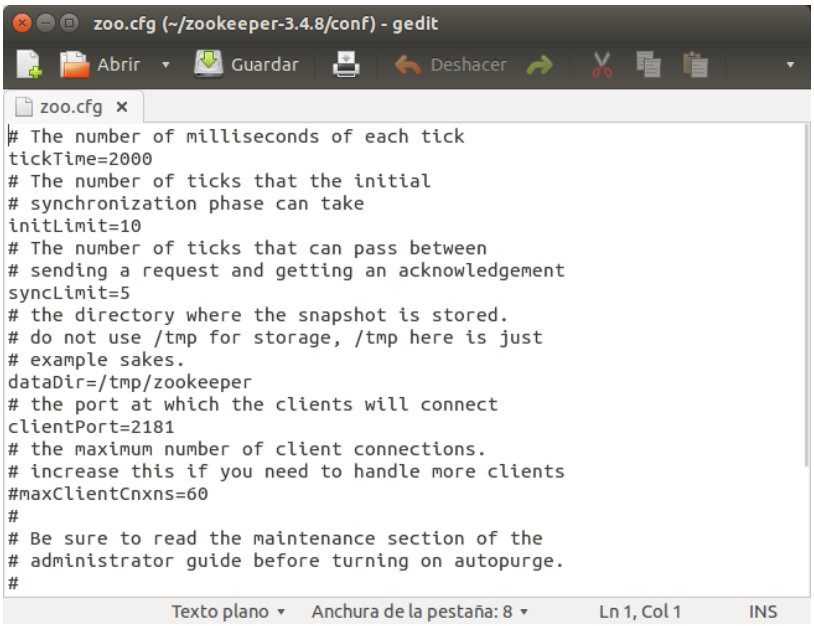


Figura 5: Archivos de configuración de Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

8. Con este comando vamos a entrar de manera segura a las siguientes archivos que tienen algún tipo de seguridad, Ya que estamos dentro de los archivos, iniciamos como root ilustrado en la figura 6.

```
sudo mkdir -p /var/lib/zookeeper
sudo su
```

```
ximena@xlsamaniego:~/zookeeper-3.4.8$ sudo mkdir -p /var/lib/zookeeper
[sudo] password for ximena:
Lo sentimos, vuelva a intentarlo.
[sudo] password for ximena:
ximena@xlsamaniego:~/zookeeper-3.4.8$ sudo su
```

Figura 6: Comando seguro para acceder a los archivos de Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

9. Ya estando del root vamos habilitar una identificación para cada uno de los nodos que vamos a utilizar es decir para el servidor 1 vamos a configurar lo siguiente también ilustrado en la figura 7.

```
echo "1" > /var/lib/zookeeper/myid
```

Habilitamos para el servidor 2

echo "2" > /var/lib/zookeeper/myid

Habilitamos para el servidor 3

echo "3" > /var/lib/zookeeper/myid

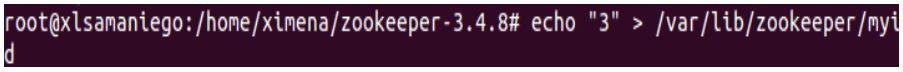


Figura 7: Comando para asignar id a cada servidor.
Elaboración: La Autora, Ximena Samaniego A.

10. Ahora vamos a iniciar Zookeeper, estando dentro del root vamos a colocar el siguiente también ilustrado en la figura 8:

bin/zkServer.sh start

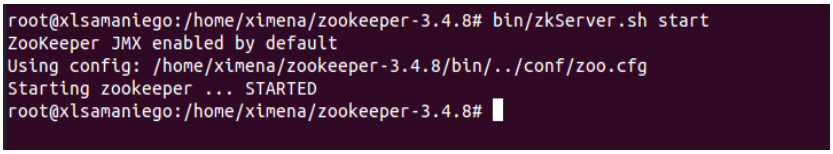


Figura 8: Comando para iniciar Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

ANEXO B: Instalación de Cassandra

Requisitos

1. Visitamos la página de www.planetcassandra.org/cassandra donde encontraremos las versiones disponibles de Cassandra para sistema operativo Linux, una vez descargado procedemos a la instalación.

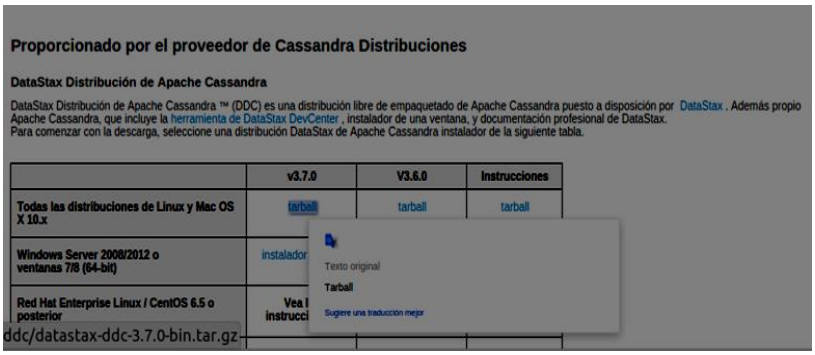


Figura 1. Página de descarga de Cassandra.
Fuente: https://academy.datastax.com/planet-cassandra/cassandra.
Elaboración: DataStax (2016)

2. Se debe contar con la versión de java instalado e actualizado es decir con jre8 con el siguiente comando revisamos la versión de java.

   java -versión

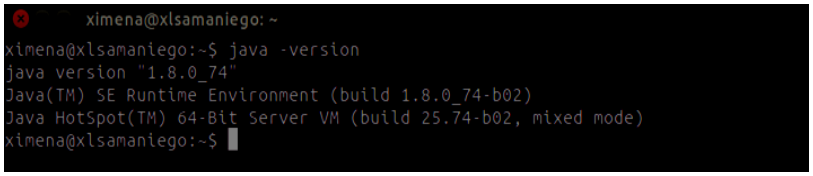


Figura 2. Versión de Java
Elaboración: propia.

3. Creamos una carpeta llamada Cassandra con el siguiente comando (**mkdir Cassandra**), después que se haya creado la carpeta vamos acceder a la misma, luego copiamos del archivo de documentos el archivo.tar de la carpeta de Cassandra.

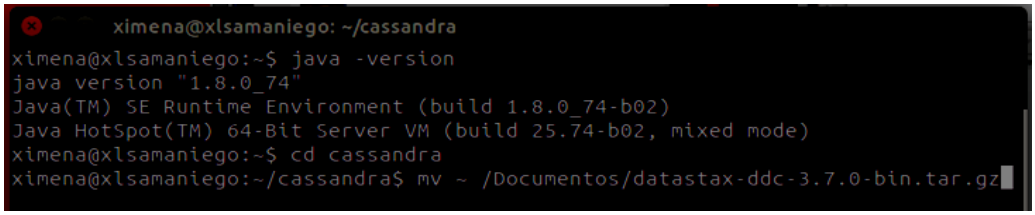


Figura 3. Creación de la carpeta Cassandra
Elaboración: propia.

4. Ahora vamos a descomprimir el archivo.tar con el comando siguiente, (**tar -xvzf datastax-3.7.0-bin.tar.gz**), luego entramos a la carpeta de datastax-ddc-3.7.0 y revisamos cada uno de los archivos de Cassandra.

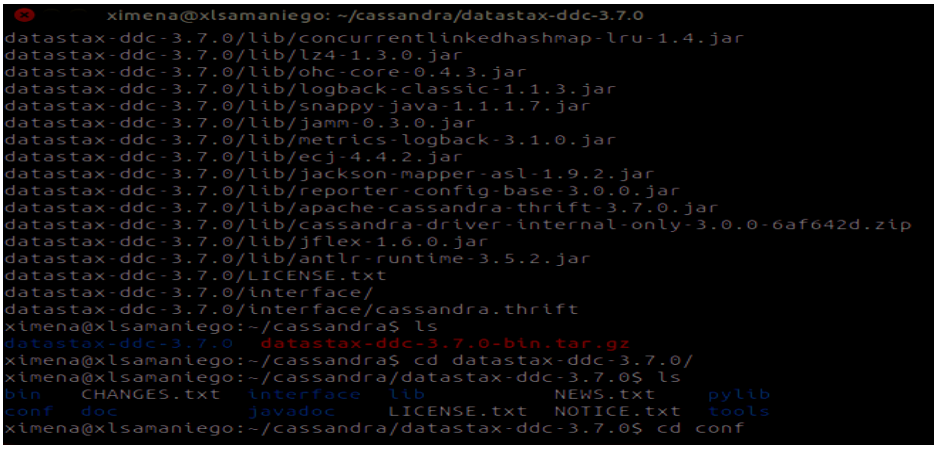


Figura 4. Descompresión de documento de Cassandra y revisión de archivos de Cassandra.
Elaboración: propia.

5. Entramos a la carpeta de conf donde encontramos archivos de configuración.

Figura 5. Lista de archivos dentro del conf.
Elaboración: propia.

6. Con el comando de gedit entramos al archivo de configuración donde vamos a realizar unas modificaciones cassandra.yaml

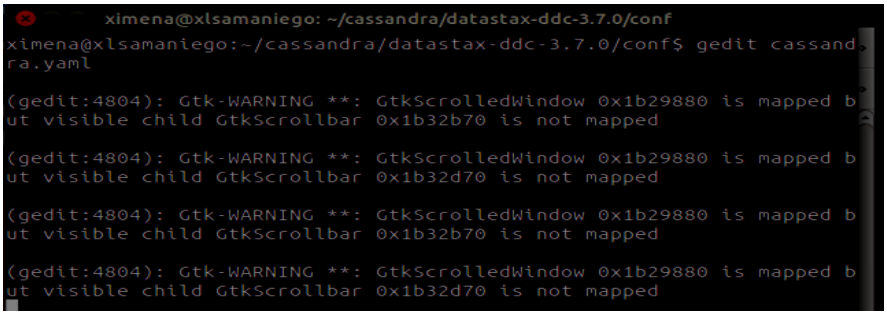


Figura 6. Modificación del archivo cassandra.yaml.
Elaboración: propia.

7. Ya que vamos a modificar en archivo de cassandra.yaml, primero vamos a modificar el nombre del clúster que se encuentre habilitado, además en número de tokens será habilitado a 256.

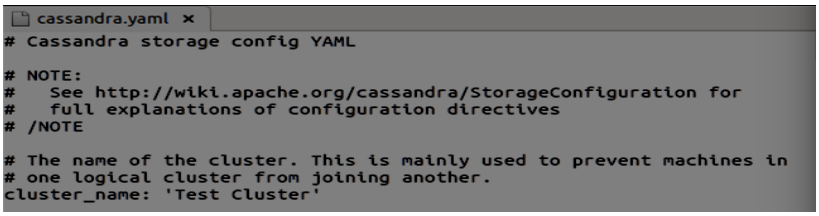


Figura 7. Habilitación del nombre del clúster.
Elaboración: propia.

8. También verificamos la partición de org.apache.cassandra.dht.Murmur3Partitioner y su directorio que es var/lib/cassandra/data

Figura 8. Verificación de la partición de Cassandra.
Elaboración: propia.

9. También verificamos la partición de org.apache.cassandra.dht.Murmur3Partitioner y su directorio que es var/lib/cassandra/data.

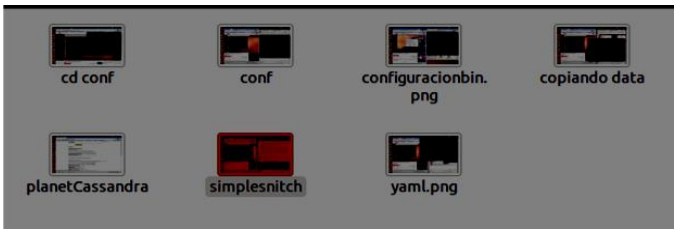


Figura 9. Verificación de la partición de Cassandra.
Elaboración: propia.

10. En el archivo de cassandra.yaml, agregamos un endpoint.snitch y se verifica que el endopoint_snitch: SimpleSnitch.

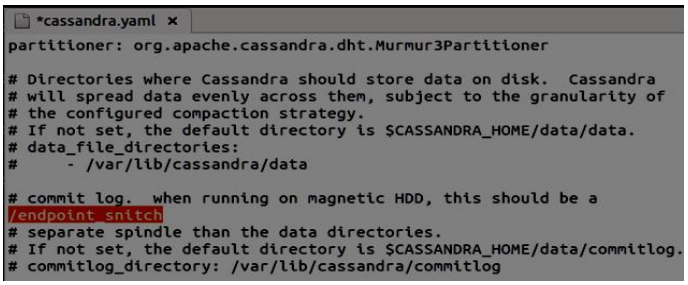


Figura 10. Verificación del endpoint_snitch.
Elaboración: propia.

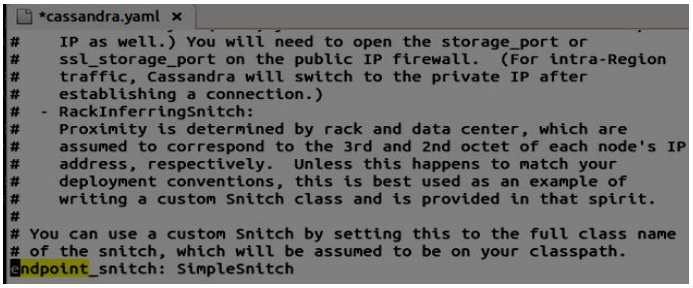


Figura 11. Verificación del endpoint_snitch: SimpleSnitch
Elaboración: propia.

11. Ahora vamos a crear nuestro segundo directorio donde estableceremos permisos de Cassandra.

Figura 12. Creación del segundo directorio de Cassandra
Elaboración: propia.

12. Damos permisos a los dos directorios creados. Iniciamos Cassandra con el siguiente comando de bin/cassandra por el cual estaríamos ejecutando en segundo plano.

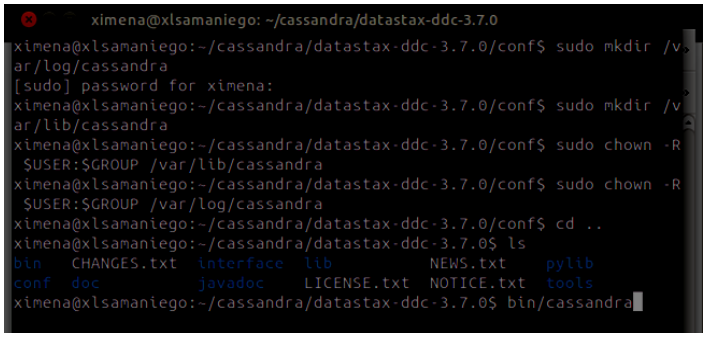


Figura 13. Permiso a directorios creados, inicio de Cassandra.
Elaboración: propia.

13. Una vez iniciado Cassandra el resultado será el siguiente.

Figura 14. Permiso a directorios creados, inicio de Cassandra.
Elaboración: propia.

14. Ya que tengamos iniciado a Cassandra vamos de otra manera a clasificar los centros en el primer plano abrimos otro terminal donde vamos a ejecutar los siguiente, (ps aux | grep cass) y damos enter.

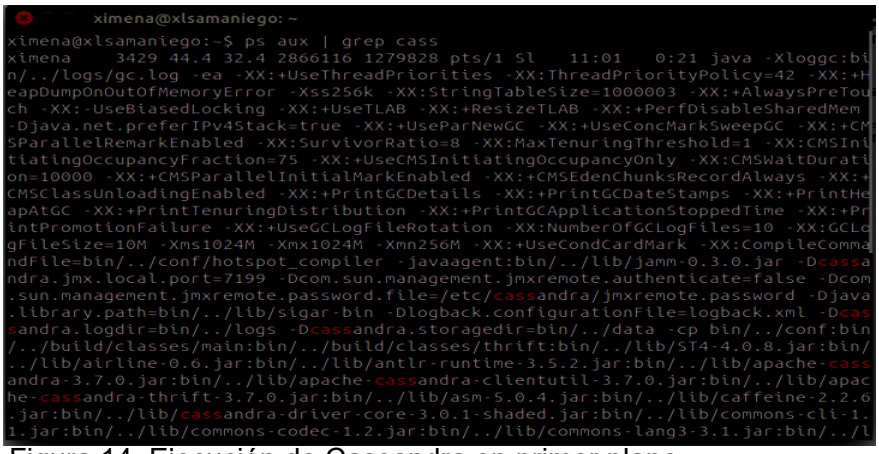


Figura 14. Ejecución de Cassandra en primer plano.
Elaboración: propia.

En esta nueva ventana, vamos a ir a buscar el id del proceso de Cassandra aquí observamos que el ID de proceso en este caso es 3429

15. En la misma ventana pero en otra línea de comando vamos a poner lo siguiente, (Kill 3429) donde se va matar el proceso de Cassandra

Figura 15. Matamos id proceso Cassandra.
Elaboración: propia.

16. Ahora vamos al terminal donde se inició Cassandra y cortamos el proceso con Ctrl+c y en otra línea ejecutamos bin/cassandra –f

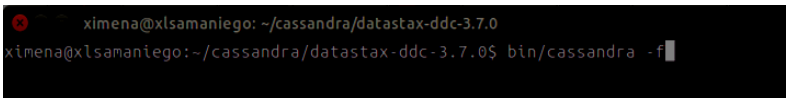


Figura 16. Matamos id proceso Cassandra.
Elaboración: propia.

17. En el terminal anterior donde colocamos el ID volvemos a escribir otro vez el siguiente comando para solicitar la vista del ID del proceso.

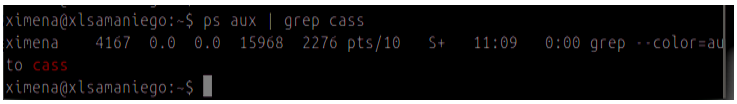


Figura 17. Buscamos Id de proceso.
Elaboración: propia.

Como podemos observar no se ha encontrado ningún identificador de proceso.

18. Ahora se comprobara el estado de internet primero debemos estar conectado a Cassandra y luego en otro terminal.

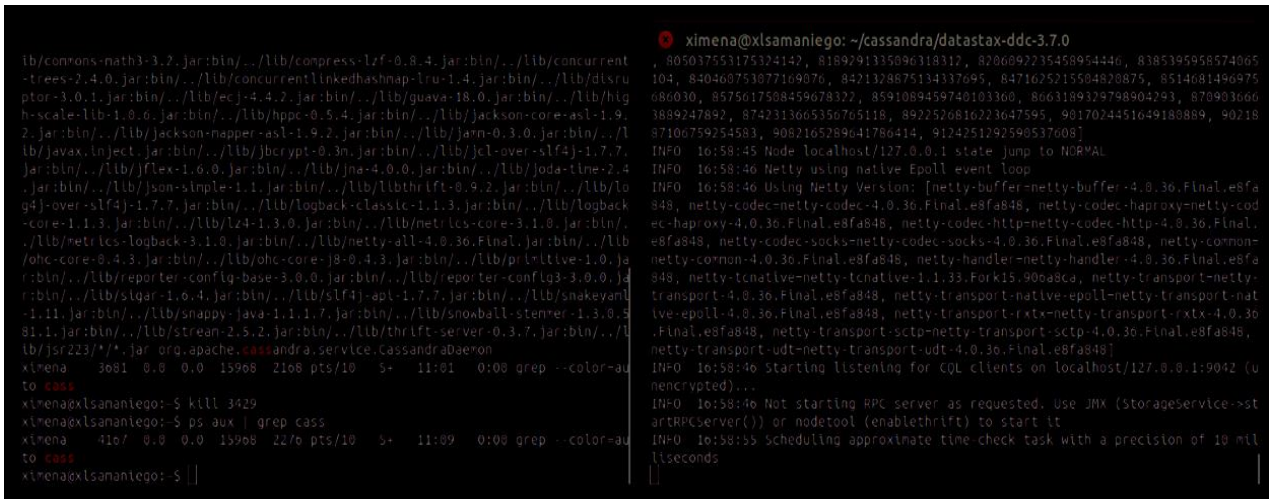


Figura 2. Ejecución de Cassandra y Búsqueda del Id proceso.
Elaboración: propia.

19. En donde se está ejecutando Cassandra colocamos lo siguiente, (cd ~/cassandra/datastax-ddc-3.7.0/),

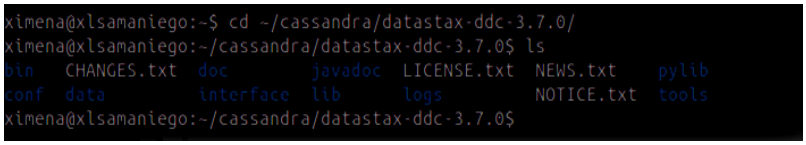


Figura 18.Creacion de Cassandra en directorio especificado.
Elaboración: propia.

20. Vamos a poder ver el estado de internet, con el siguiente comando (bin /nodetool status).

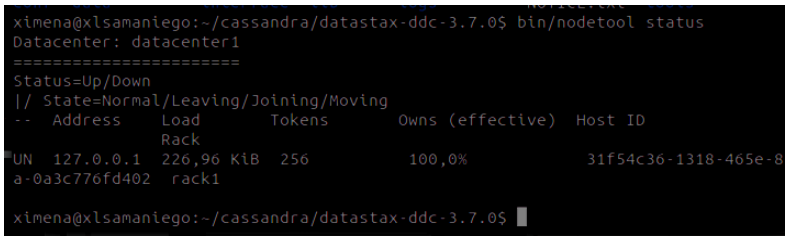


Figura 19.Estado de Internet.
Elaboración: propia.

Aquí nos muestra un centro de datos, donde se inició correctamente el nodo.

21. Por ultimo vamos a colocar este comando (bin/nodetool ring). Nos muestra el estado de nodo y la información sobre el anillo según lo determinado por el nodo que se está consultando. Esta información le puede dar una idea del equilibrio de carga y si los nodos están abajo. Si el clúster no está configurado correctamente, diferentes nodos pueden mostrar un anillo diferente.

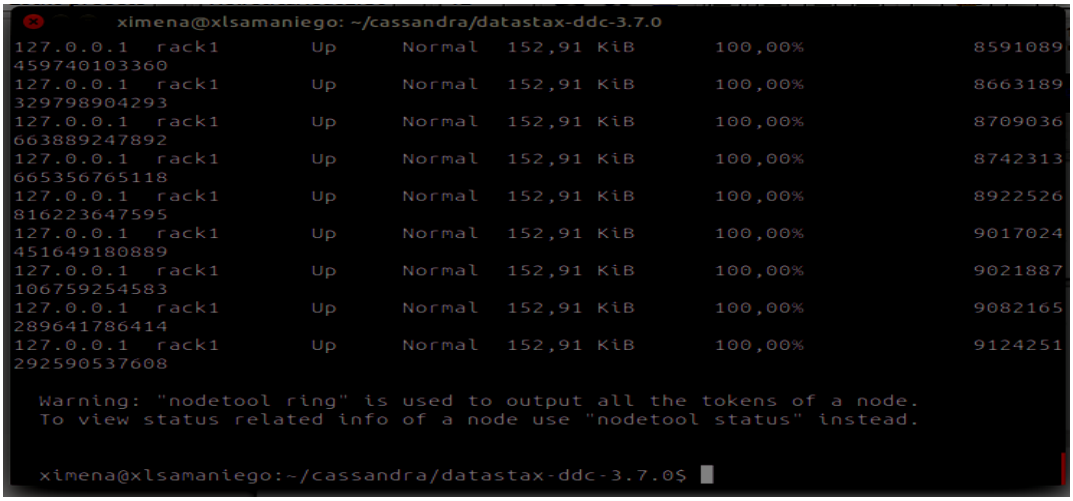


Figura 20.Estado del nodo.
Elaboración: propia.

ANEXO C: Instalación de sistema de monitoreo DataDog

1. Hay que tener en cuenta que ya debemos tener instalado los sistemas Distribuidos tanto de Zookeeper como de Cassandra.
2. Nos dirigimos a la página de DataDog y nos vamos a registrar.

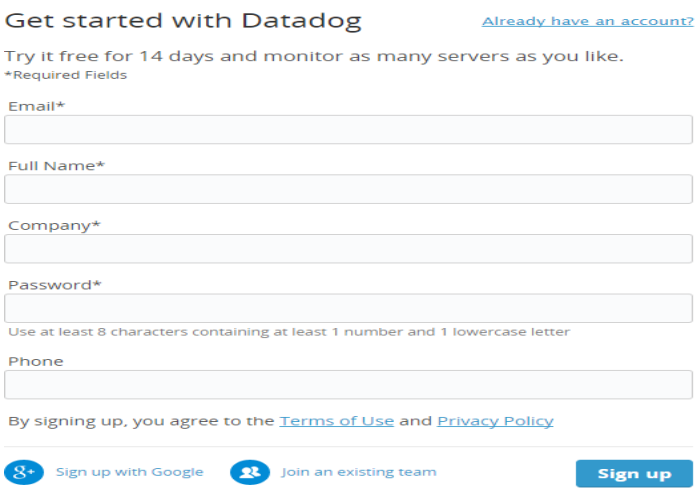


Figura 1. Registro de Datadog
Elaboración: La Autora, Ximena Samaniego A.

Una vez que tengamos nuestra cuenta de prueba de DataDog vamos a realizar un análisis estadístico tanto de Zookeeper como Cassandra de acuerdo a sus métricas.

ANEXO D: Instalación de Zookeeper en sistema operativo Windows

1. Nos dirigimos a la página de apache Zookeeper y descargamos la versión estable que está disponible.

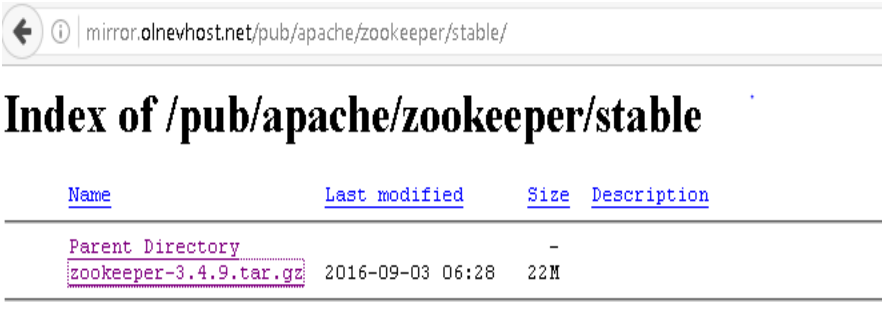


Figura 1: Página de descargar Zookeeper.

Elaboración: La Autora, Ximena Samaniego A.

2. Vamos crear las siguientes capetas y a guardarlo en el c: \dev \tools.

Figura 2: Directorio de Zookeeper

Elaboración: La Autora, Ximena Samaniego A.

3. Establecer las variables de entorno.

- Para ellos vamos a ir a las variables de sistema donde se va modificar variables de entorno para java.
- Vamos a crear una variable con JAVA_HOME con la ruta donde se encuentra el jdk como a continuación.

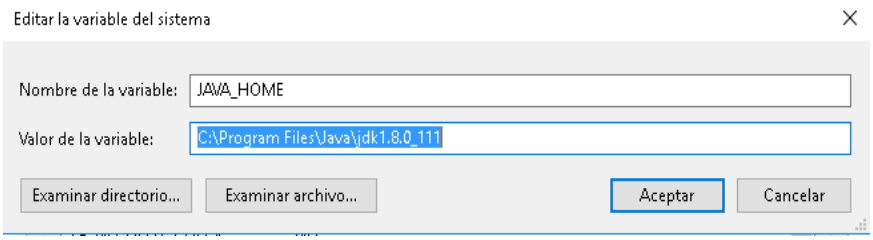


Figura 3: Creación de variable de entorno java

Elaboración: La Autora, Ximena Samaniego A.

- Ahora vamos a editar el path donde vamos añadir %JAVA_HOME%\bin, una vez hecho esto damos aceptar.

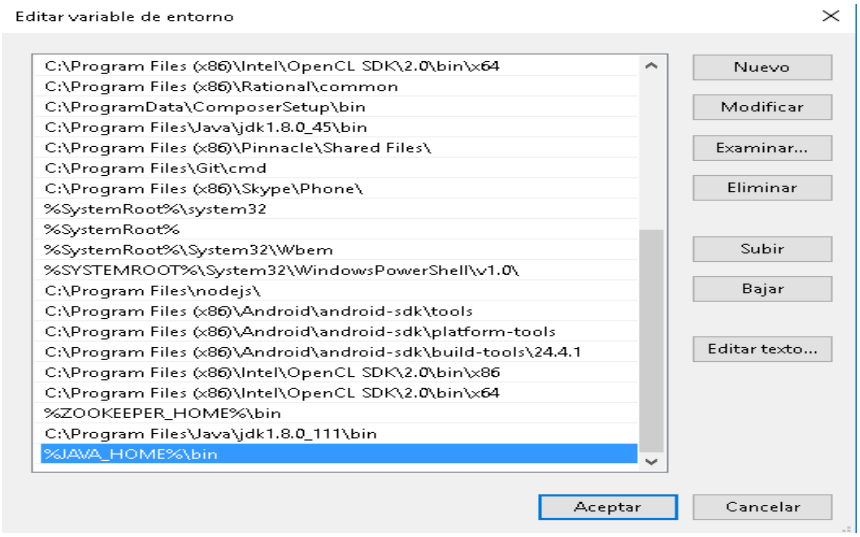


Figura 4: Modificación de variable de entorno java

Elaboración: La Autora, Ximena Samaniego A.

- Ahora vamos a crear una variable con ZOOKEEPER_HOME con la ruta donde se encuentra Zookeeper como a continuación.

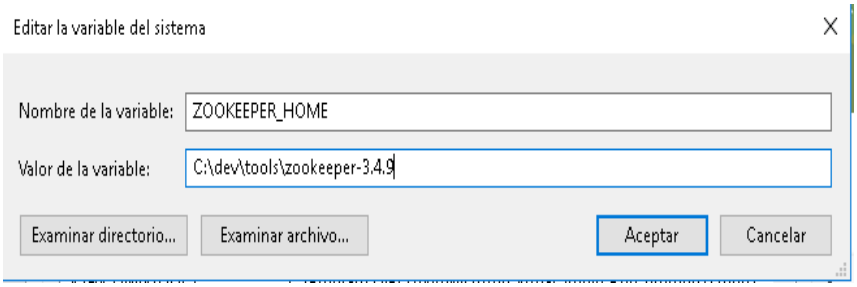


Figura 5: Creación de variable de entorno Zookeeper

Elaboración: La Autora, Ximena Samaniego A.

- Vamos a editar el path donde vamos añadir %ZOOKEEPER_HOME%\bin, damos aceptar y continuamos.

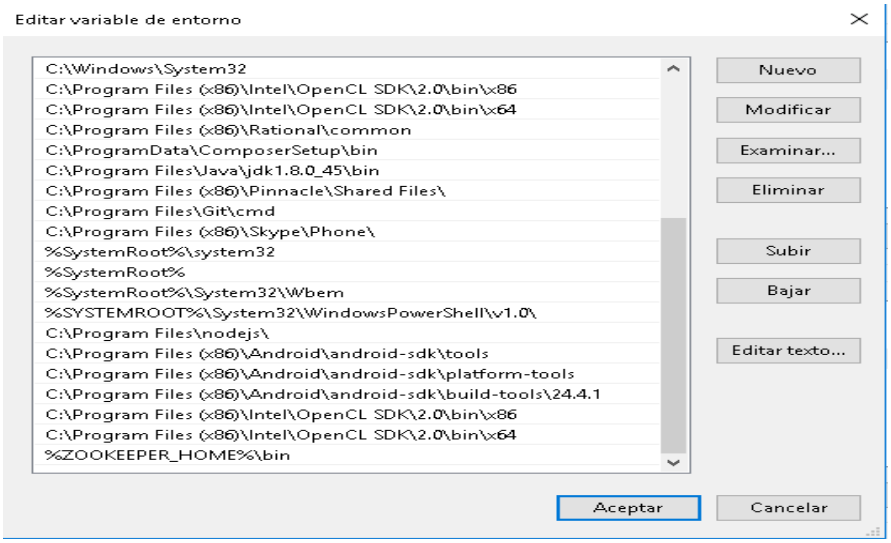


Figura 6: Modificación de variable de entorno Zookeeper

Elaboración: La Autora, Ximena Samaniego A.

4. Configuración del servidor Zookeeper.

- Nos dirigimos a la carpeta de Zookeeper en el archivo conf vamos a realizar una copia de zoo_sample.cfg y colocamos el nombre de zoo.cfg como a continuación.

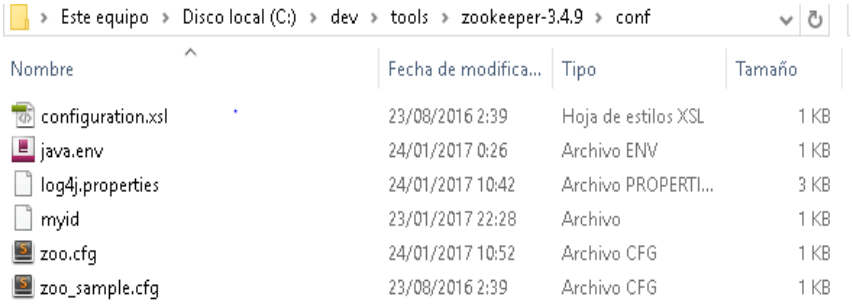


Figura 7: Ruta de archivo conf de Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

- Editamos el archivo conf de Zookeeper de la siguiente manera y guardamos.

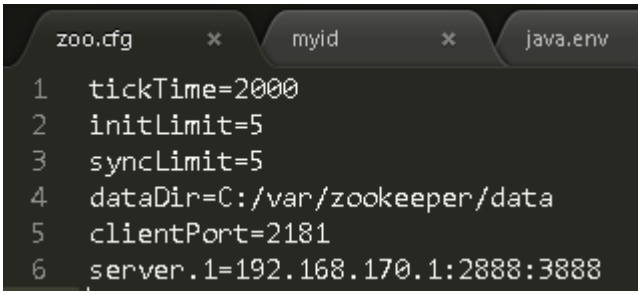


Figura 8: Ruta de archivo conf de Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

- En siguiente paso es crear un myid, si se fijan en el anterior archivo de zoo.cfg se escribió en el dataDir una ruta de data, lo que significa que vamos a crear ese directorio en la unidad c donde vamos a crear un archivo de myid donde servirá para identificar la instancia de Zookeeper.

Figura 9: Ruta de archivo myid de Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

5. Prueba de Ejecución de servidor de Zookeeper

- Vamos a abrir el command prompt de Windows y vamos a ejecutar el siguiente comando de Zookeeper para que inicie el servidor de Zookeeper zkServer.cmd.

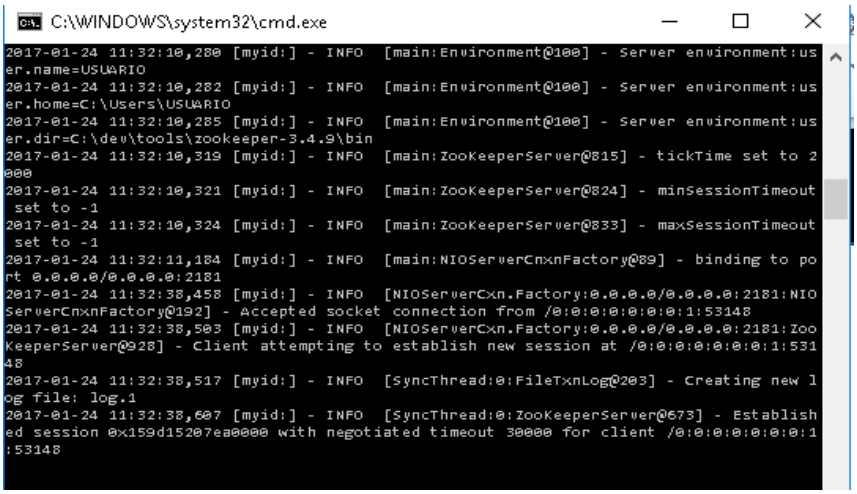


Figura 10: Inicio de servidor Zookeeper.

Elaboración: La Autora, Ximena Samaniego A.

- En otra ventana de comando vamos ejecutar el cliente con zkCli.cmd donde va ejecutar Zookeeper.

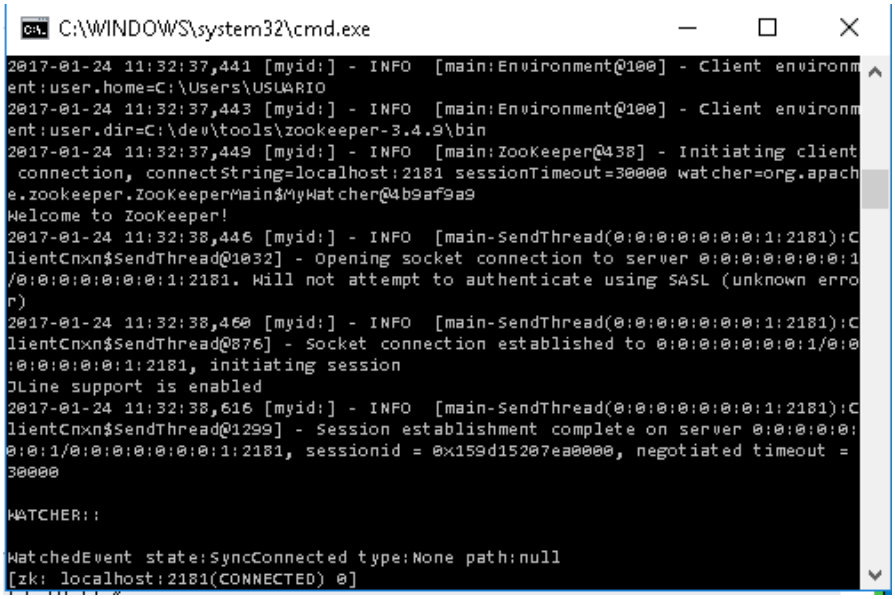


Figura 11: Ejecución de servidor Zookeeper.
Elaboración: La Autora, Ximena Samaniego A.

ANEXO E: Funcionamiento de Zookeeper Cli

1. Con ZooKeeper instalado y todas sus configuraciones procedemos a que todos los 3 nodos se encuentren en una misma red y se pueda hacer los ping entre ellas.
2. Hay que tomar en cuenta la dirección ip que nos asigna a las maquinas o nodos.
3. Iniciamos ZooKeeper en cada uno de las maquinas como lo demostrado en la figura

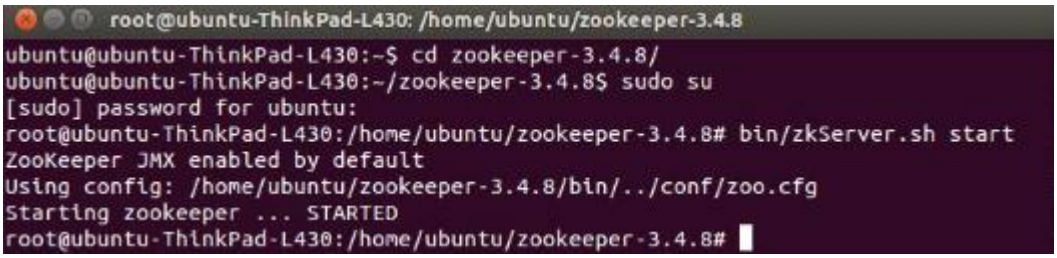


Figura 1. Comando para iniciar Zookeeper
Elaboración: La Autora, Ximena Samaniego A.

4. Para saber que Zookeeper está corriendo perfectamente colocamos el siguiente comando ya que este inicializado Zookeeper como se demuestra en la figura 2.

   bin/zkServer.sh status

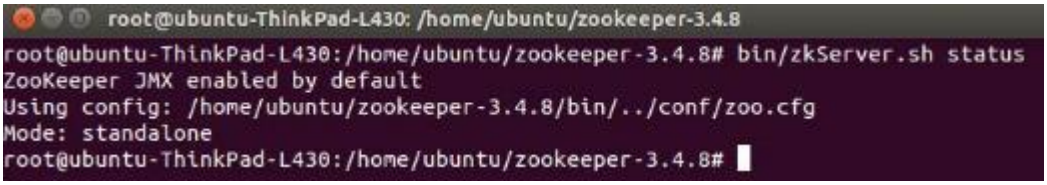


Figura 2**.** Comando para ver el estado de Zookeeper
Elaboración: La Autora, Ximena Samaniego A.

5. Ahora vamos a comunicar las máquinas de acuerdo a la dirección ip de cada uno de los nodos en nuestro caso.

   Nodo 1 = 172.18.4.187: 2181
   Nodo 2 = 172.18.4.179: 2181
   Nodo 3 = 172.18.4.195: 2181

   A continuación procedemos a iniciar la librería de ZooKeeper Cli más la dirección ip de la maquina principal en nuestro caso el nodo 1 y se continúa con la conexión del cliente en las 3 máquinas con el comando siguiente:

bin/zkCli.sh –server 172.18.4.187

Conexión del nodo 1:

```
root@ubuntu-ThinkPad-L430:/home/ubuntu/zookeeper-3.4.8# bin/zkCli.sh -server 172
.18.4.187:2181
Connecting to 172.18.4.187:2181
```

Figura 3. Comando para el nodo 1 y conexión con el nodo principal mediante la librería de cliente.
Elaboración: La Autora, Ximena Samaniego A.

```
Welcome to ZooKeeper!
2016-05-18 10:31:45,032 [myid:] - INFO  [main-SendThread(172.18.4.187:2181):Clie
ntCnxn$SendThread@1032] - Opening socket connection to server 172.18.4.187/172.1
8.4.187:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2016-05-18 10:31:45,095 [myid:] - INFO  [main-SendThread(172.18.4.187:2181):Clie
ntCnxn$SendThread@876] - Socket connection established to 172.18.4.187/172.18.4.
187:2181, initiating session
[zk: 172.18.4.187:2181(CONNECTING) 0] 2016-05-18 10:31:45,111 [myid:] - INFO  [m
ain-SendThread(172.18.4.187:2181):ClientCnxn$SendThread@1299] - Session establis
hment complete on server 172.18.4.187/172.18.4.187:2181, sessionid = 0x154c46eed
e60001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

[zk: 172.18.4.187:2181(CONNECTED) 0]
```

Figura 17. Shell de Zookeeper de conexión exitosa nodo 1
Elaboración**:** La Autora, Ximena Samaniego A.

Conexión del nodo 2:

```
root@ubuntu-ThinkPad-L430:/home/ubuntu/zookeeper-3.4.8# bin/zkCli.sh -server 172
.18.4.187:2181
Connecting to 172.18.4.187:2181
2016-05-18 10:32:10,710 [myid:] - INFO  [main:Environment@100] - Client environm
ent:zookeeper.version=3.4.8--1, built on 02/06/2016 03:18 GMT
2016-05-18 10:32:10,714 [myid:] - INFO  [main:Environment@100] - Client environm
ent:host.name=ubuntu-ThinkPad-L430
2016-05-18 10:32:10,714 [myid:] - INFO  [main:Environment@100] - Client environm
ent:java.version=1.8.0_74
2016-05-18 10:32:10,716 [myid:] - INFO  [main:Environment@100] - Client environm
ent:java.vendor=Oracle Corporation
2016-05-18 10:32:10,716 [myid:] - INFO  [main:Environment@100] - Client environm
```

Figura 4. Comando para el nodo 2 y conexión con el nodo principal mediante la librería de cliente.
Elaboración: La Autora, Ximena Samaniego A.

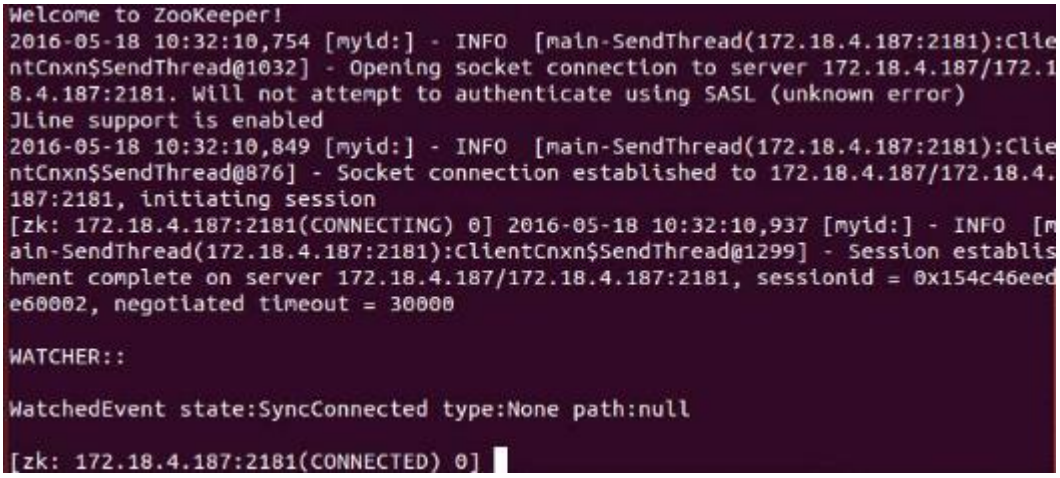


Figura 5. Shell de Zookeeper de conexión exitosa nodo 2
Elaboración: La Autora, Ximena Samaniego A.

Conexión del nodo 3:

6. Para llevar a cabo las operaciones del CLI, el servidor ZooKeeper debe estar iniciado y a continuación el cliente ZooKeeper ("bin / zkCli.sh"). Una vez que el cliente se inicia, puede realizar la siguiente operaciones:

- Crear znodes
- Obtener datos
- Ver znodes
- Creación de subznodes
- Listar znodes
- Comprobar estado
- Eliminar un znode

Ahora vamos a probar cada una de las funciones de ZCli.

6.1 Crear Znodes

Crear un znode con la ruta dada específica si el znode creado será efímera, persistente o secuencial. Por defecto, todos los znodes son persistentes.

Znodes efímeras: Se eliminarán de forma automática cuando una sesión de vencimiento o cuando el cliente se desconecta.

Znodes secuenciales: Garantiza de que el camino znode será único.

6.1.1. Ahora vamos a crear el primer nodo el cual se llamara FirstZnode desde el Shell de Zookeeper:

```
[zk: 172.18.4.187:2181(CONNECTED) 0] create /FirstZnode "MyfirstZookeeper-app"
Created /FirstZnode
```

Figura 6. Creación de Znode “FirstZnode”, en la maquina principal.
Elaboración**:** La Autora, Ximena Samaniego A.

6.1.2 Verificamos que este creado el znode “FirstZnode” en las tres máquinas o nodos con el siguiente comando desde el Shell de Zookeeper.

Equipo 1:

```
[zk: 172.18.4.187:2181(CONNECTED) 2] ls /
[zookeeper, FirstZnode]
```

Figura 7. Verificación de Znode creado en la máquina principal.
Elaboración: La Autora, Ximena Samaniego A.

Equipo 2:

```
[zk: 172.18.4.187:2181(CONNECTED) 0] ls /
[zookeeper, FirstZnode]
[zk: 172.18.4.187:2181(CONNECTED) 1]
```

Figura 8. Verificación de Znode creado en la maquina 2.
Elaboración**:** La Autora, Ximena Samaniego A.

Equipo 3:

```
[zk: 172.18.4.187:2181(CONNECTED) 0] ls /
[zookeeper, FirstZnode]
```

Figura 9. Verificación de Znode creado en la maquina 3.
Elaboración: La Autora, Ximena Samaniego A.

6.1.3 Ahora vamos a crear un nodo Znode secuencial en la maquina principal el cual se va añadir –s.

```
[zk: 172.18.4.187:2181(CONNECTED) 3] create -s /FirstZnode “second-data”
Created /FirstZnode0000000001
```

Figura 10. Creación de Znode secuencial, en la maquina principal.
Elaboración: La Autora, Ximena Samaniego A.

Para verificar que se haya creado correctamente colocamos el siguiente comando en las tres máquinas como en la figura 11.

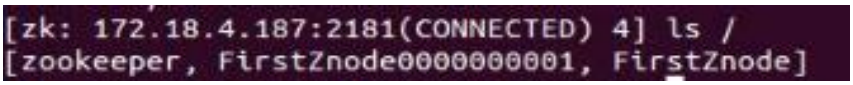


Figura 11. Verificación de znode creado en los 3 equipos.
Elaboración: La Autora, Ximena Samaniego A.

6.1.4 Creación de un Znode efímero añadimos – e en el comando siguiente.

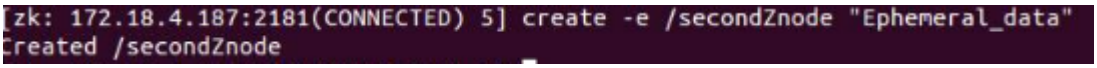


Figura 12. Creación de znode efímero creado en los 3 equipos.
Elaboración: La Autora, Ximena Samaniego A.

Figura 13. Verificación de znode efímero creado en los 3 equipos.
Elaboración: La Autora, Ximena Samaniego A.

Recuerde que cuando se pierde una conexión de cliente, se eliminará el znode efímera. Puedes probarlo al dejar el ZooKeeper CLI y luego volver a abrir la CLI como lo demostrado en figura 14.

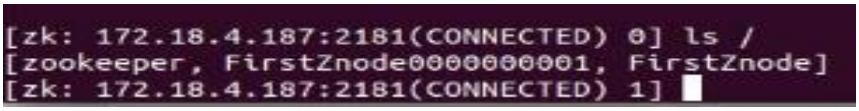


Figura 14. Verificación de eliminación znode efímero creado en los 3 equipos.
Elaboración**:** La Autora, Ximena Samaniego A.

6.2 Obtener Datos

Devuelve los datos asociados de la znode y los metadatos de la znode especificado. Obtendrá información tal como cuando los datos de la última modificación, en el que se modificó, y la información sobre los datos. Este CLI también se utiliza para asignar relojes para mostrar notificación acerca de los datos.

A continuación vamos a obtener un Znode creado como “FirstZnode” se lo deberá obtener en los tres equipos.

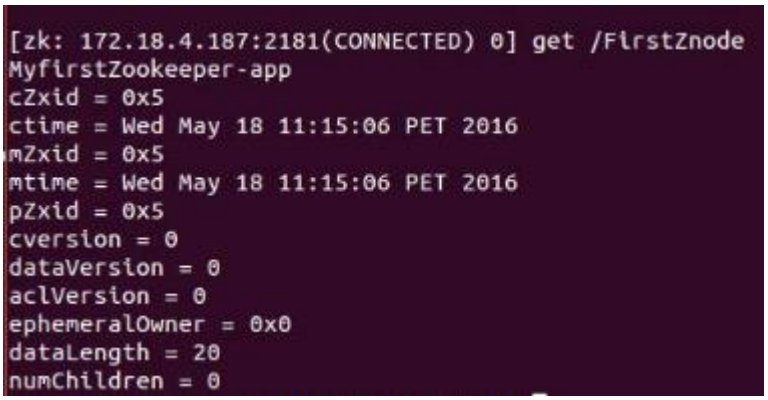


Figura 15. Obtención de Znode “FirstZnode”.
Elaboración: La Autora, Ximena Samaniego A.

Para acceder a un znode secuencial, debe introducir la ruta completa de la znode esto se debe presentar en las 3 máquinas.

Figura 16. Obtención de Znode secuencial “FirstZnode0000000003”.
Elaboración: La Autora, Ximena Samaniego A.

6.3 Reloj

Los relojes muestran una notificación cuando cambian los datos de los znode. Se puede establecer un reloj sólo con el comando get.

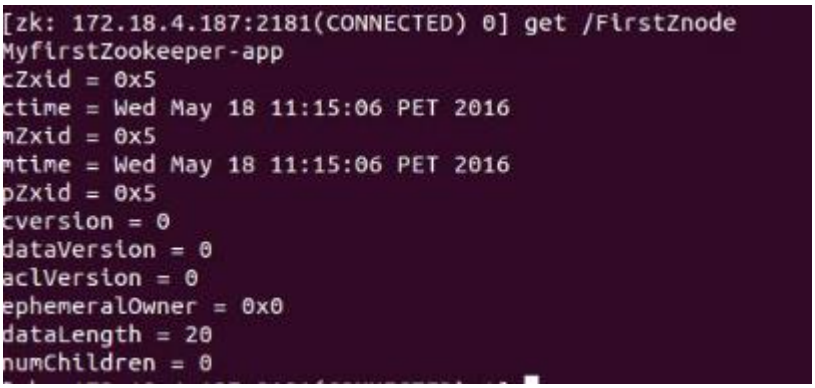


Figura 17. Obtención de Znode FirstZnode.
Elaboración: La Autora, Ximena Samaniego A.

6.4 Conjunto de datos

Ajuste los datos de la znode especificado. Una vez que termine esta operación dada, se puede comprobar los datos utilizando el **set** comando CLI.

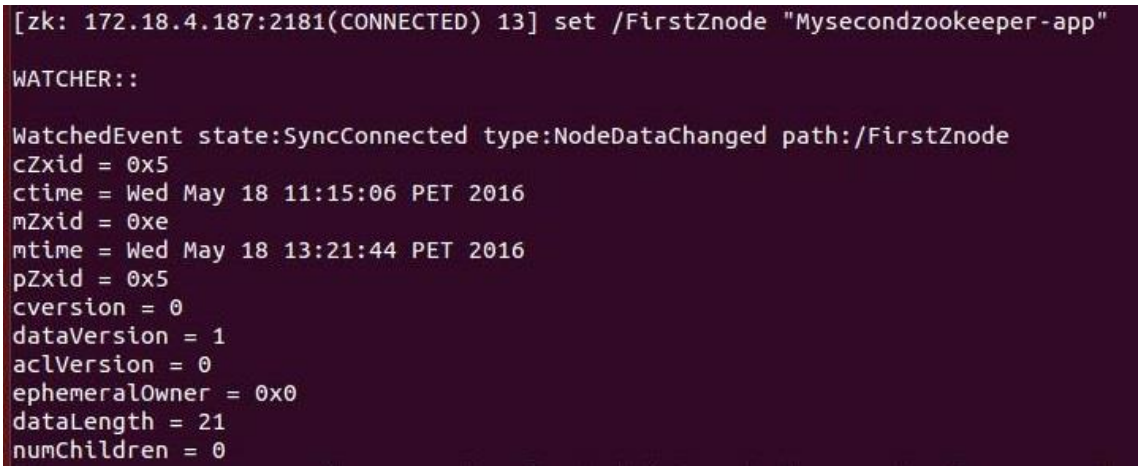


Figura 18. Modificación de datos del SecondZnode.
Elaboración: La Autora, Ximena Samaniego A.

6.5 Creación de subznodes

Creación de los subznodes es similar a la creación de nuevas znodes. La única diferencia de un znode es que un subznodes tendrá una ruta padre del znode.

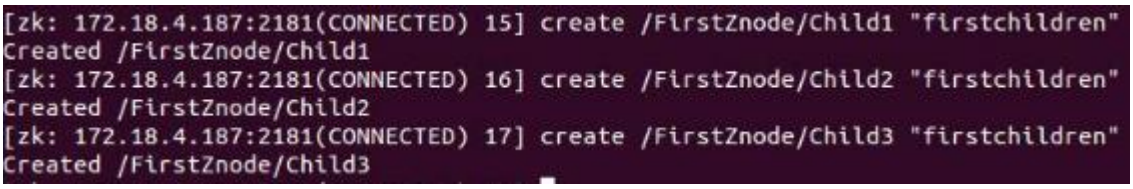


Figura 19. Creación de nodos con sus debidos SubZnodes.
Elaboración: La Autora, Ximena Samaniego A.g

6.6 Lista SubZnodes

Este comando se utiliza para enumerar y mostrar los hijos de un znode.

Figura 20. Vista de los SubZnodes.
Elaboración**:** La Autora, Ximena Samaniego A.

6.7 Comprobación de estados

El comando de status describe los metadatos de un znode especificado. Contiene detalles tales como fecha y hora, número de versión, ACL, longitud de datos, y subznodes.

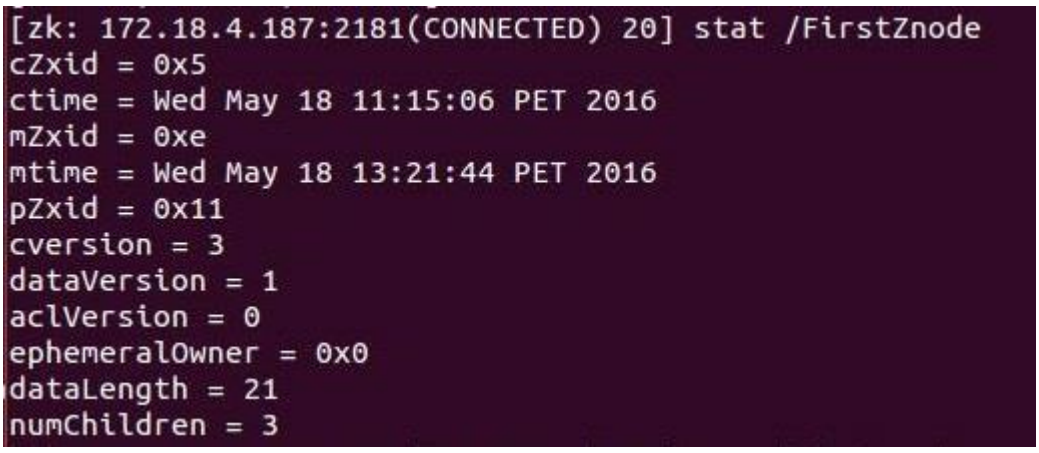


Figura 21. Comprobación de un Znode especificado.
Elaboración**:** La Autora, Ximena Samaniego A.

6.8 Retirar znode

Eliminan un znode especificado y de forma recursiva todos sus hijos. Esto sucede sólo si un znode adecuada esté disponible.

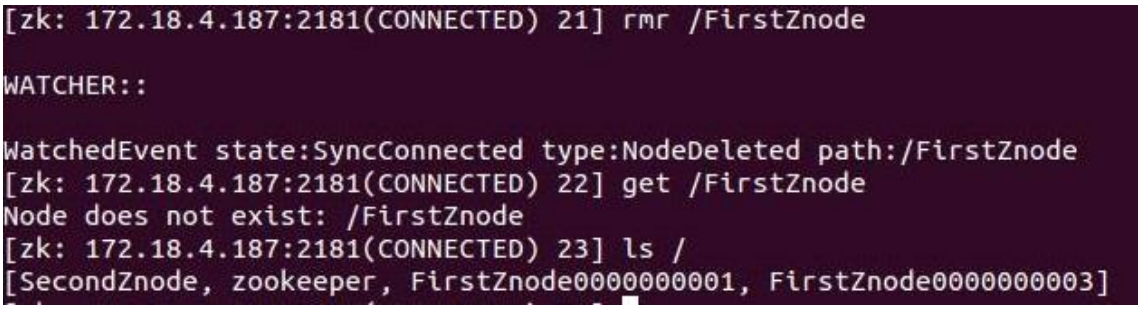


Figura 22. Eliminación de un Znode y recursivamente sus SubZnodes.
Elaboración**:** La Autora, Ximena Samaniego A.

ANEXO F: Funcionamiento de Zookeeper en Windows.

1. Creacion de znodes.

Creación de znode llamado “Firstzone1”.

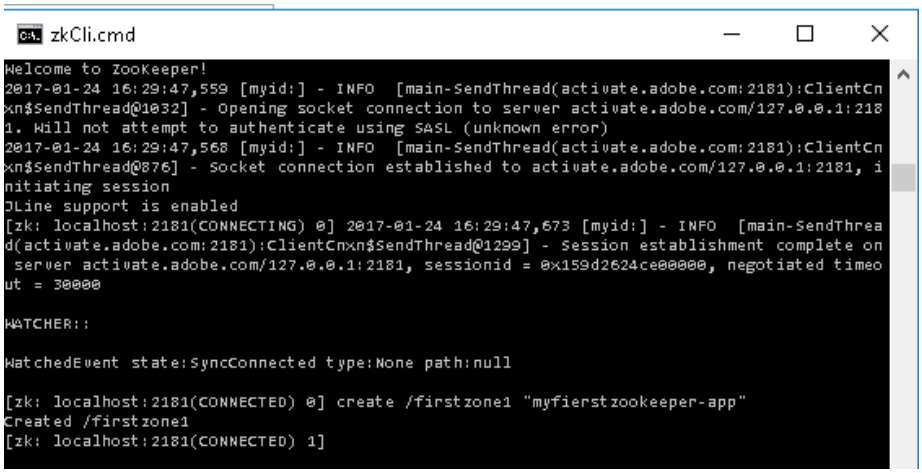


Figura 1: Creación de znode en Windows. .
Elaboración: La Autora, Ximena Samaniego A.

Creación de znode secuencial añadir –s al momento de crear nodo secuencial.

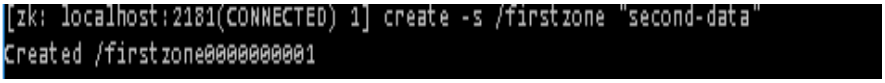
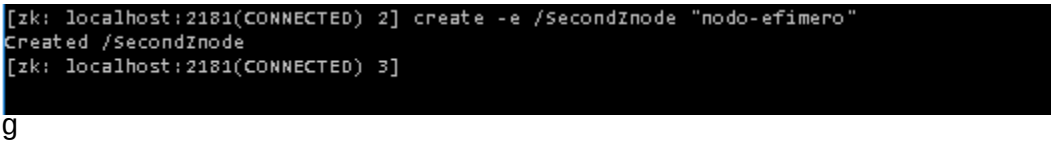


Figura 2: Creación de znode secuencial en Windows. . Elaboración: La Autora, Ximena Samaniego A.

Para crear un znode efímero añadir la opción –e como se muestra a continuación en la figura 3, cabe recalcar que si se pierde conexión de cliente, se eliminara el znode efímera.



g
figura 3: Creación de znode efímero en Windows. .
Elaboración: La Autora, Ximena Samaniego A.

Obtener datos.

Devuelve los datos asociados de la znode, obtendrá información sobre los datos que han tenido alguna modificación, también permite asignar relojes para mostrar notificación de los datos.

Obtener información del primer nodo creado.

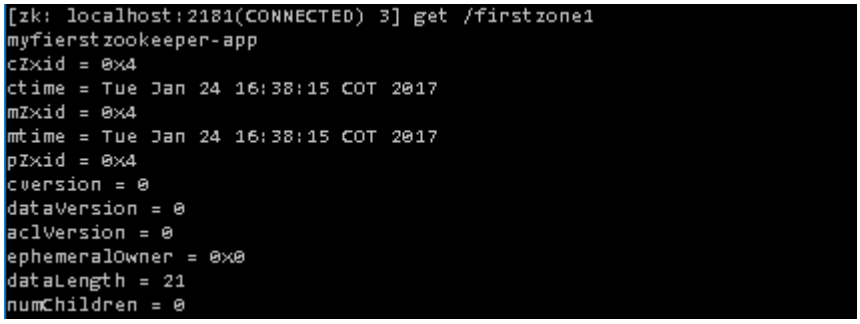


Figura 4: obtención de primer znode creado en Windows. . Elaboración: La Autora, Ximena Samaniego A.

Para acceder a un znode secuencial, debe introducir la ruta completa de la znode.

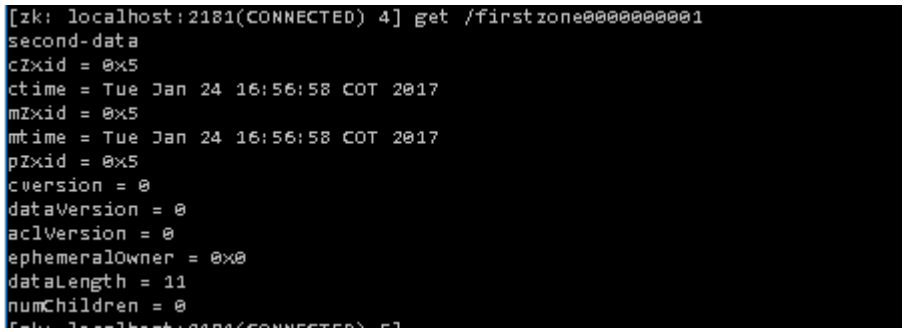


Figura 5: Obtención de znode secuencial en Windows. .
Elaboración: La Autora, Ximena Samaniego A.

2. **Reloj.**

Muestran una notificación cuando cambian los datos de subznodes (hijos)o algún znode especificada, se establece un reloj solo con el comando get.

3.1 obtenemos información del primer nodo creado con reloj, espera que haya cambios en la creación del znode.

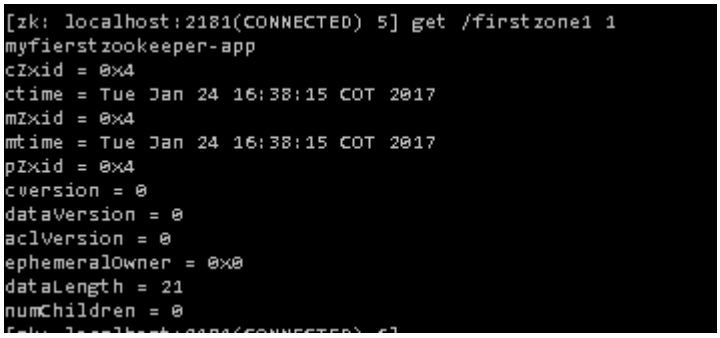


Figura 6: Obtención de reloj del primer znode en Windows. .
Elaboración: La Autora, Ximena Samaniego A.

3. Conjunto de datos.

Ajuste de datos znode especificado, una vez terminado esta operación dada, se pude comprobar los datos utilizando el comando get cli.

Si la opción de reloj asignado en el comando get, su salida será similar solo con un mensaje nuevo actualizado.

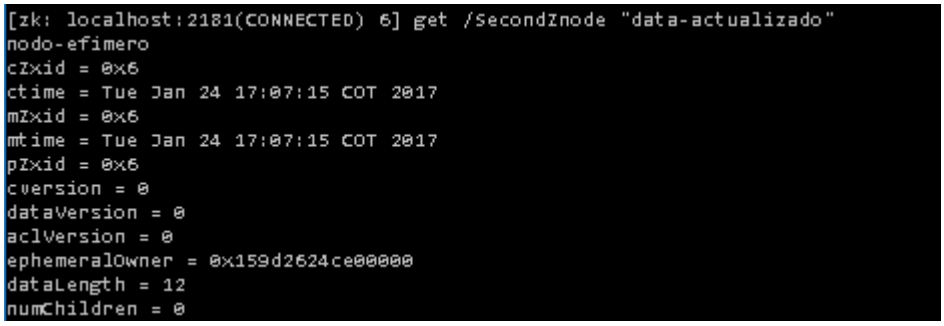


Figura 7: Obtención de conjunto de datos del primer znode en Windows.
Elaboración: La Autora, Ximena Samaniego A.

4. Creación de subznodes (hijos).

Creación de los subznodes es similar a la creación de nuevas znodes, la única diferencia es que el camino de la znode hijo tendrá otra ruta padre también.

Creación de subznodes del primer znode creado.

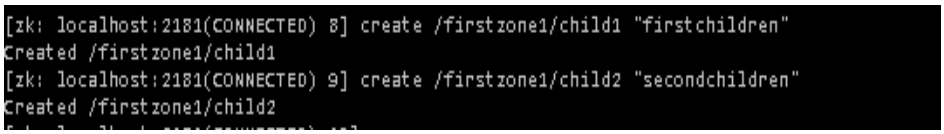


Figura 8: Creación de subznodes del primer znode en Windows.
Elaboración: La Autora, Ximena Samaniego A.

5. Listar subznodes (hijos)

Con el comando siguiente vamos a poder enumerar y mostrar los hijos de un znode.

Vamos a visualizar la creación de los subznode del primer znode creado.

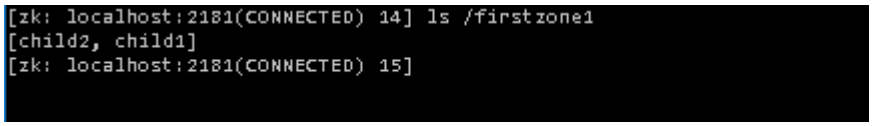


Figura 9: Visualización de subznodes del primer znode en Windows.
Elaboración: La Autora, Ximena Samaniego A.

6. Comprobación de estado.

Describe el estado de un znode especificado, contiene detalles como fecha hora, numero de versión, acl, longitud de datos y znode hijos.

Con el comando stat vamos a visualizar con más detalle información del nodo.

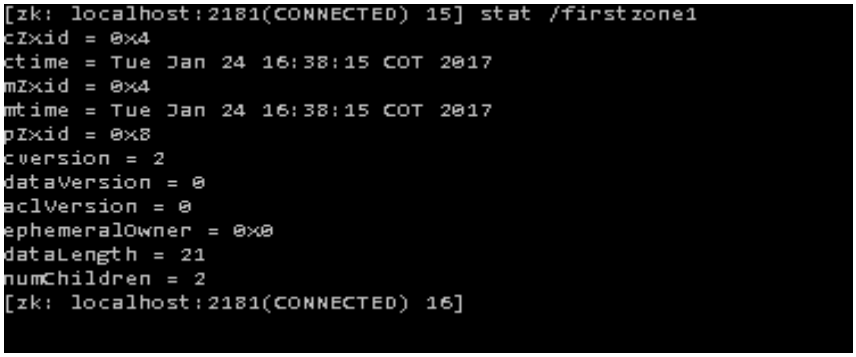


Figura 10: comprobación de estado de znode hijos.
Elaboración: La Autora, Ximena Samaniego A.

7. Retirar un znode.

Elimina un znode especificado y de igual manera a sus subznodes.

A continuación creamos un nodo de prueba y lo retiramos y por ultimo verificamos si existe.

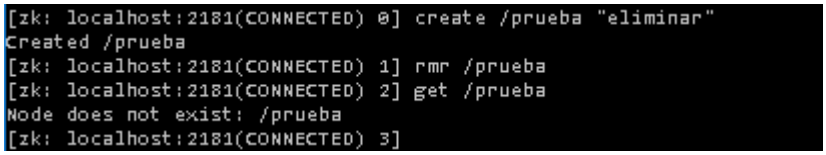


Figura 11: Eliminación de un znode.
Elaboración: La Autora, Ximena Samaniego A.

ANEXO G: Funcionamiento de Cassandra.

1. Una vez iniciado Cassandra el resultado será el siguiente.

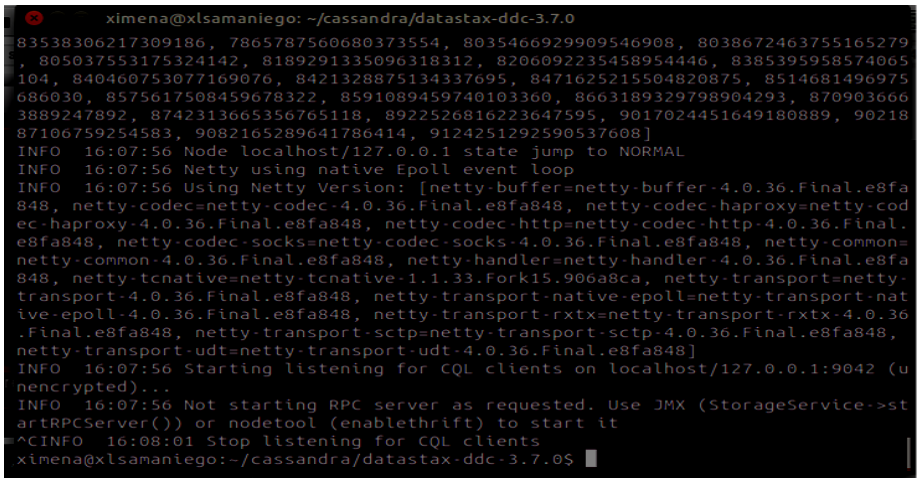


Figura 1. Permiso a directorios creados, inicio de Cassandra.
Elaboración: propia.

2. Ya que tengamos iniciado a Cassandra vamos de otra manera a clasificar los centros en el primer plano abrimos otro terminal donde vamos a ejecutar los siguiente, (ps aux | grep cass) y damos enter.

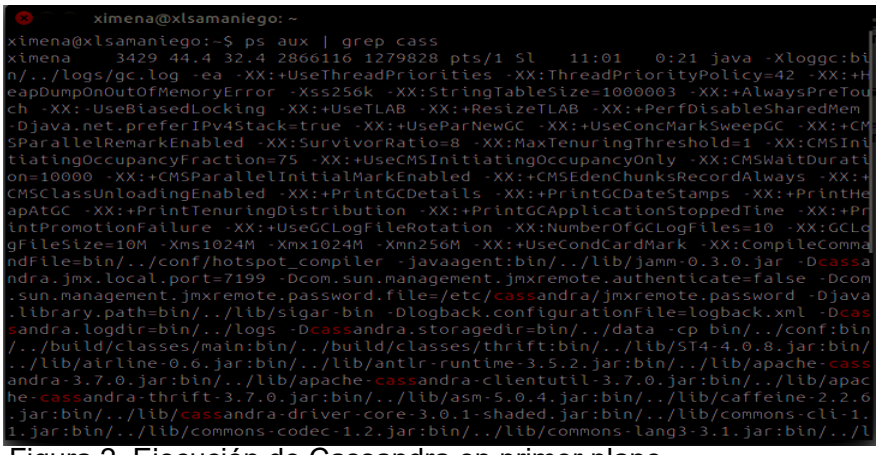


Figura 2. Ejecución de Cassandra en primer plano.
Elaboración: propia.

En esta nueva ventana, vamos a ir a buscar el id del proceso de Cassandra aquí observamos que el ID de proceso en este caso es 3429

3. En la misma ventana pero en otra línea de comando vamos a poner lo siguiente, (Kill 3429) donde se va matar el proceso de Cassandra

Figura 3. Matamos id proceso Cassandra.
Elaboración: propia.

4. Ahora vamos al terminal donde se inició Cassandra y cortamos el proceso con Ctrl+c y en otra línea ejecutamos bin/cassandra –f

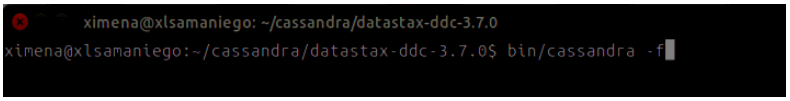


Figura 4. Matamos id proceso Cassandra.
Elaboración: propia.

5. En el terminal anterior donde colocamos el ID volvemos a escribir otro vez el siguiente comando para solicitar la vista del ID del proceso.

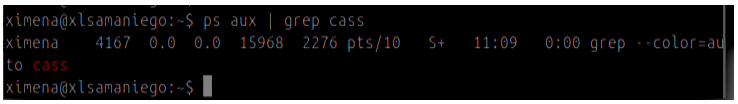


Figura 5. Buscamos Id de proceso.
Elaboración: propia.

Como podemos observar no se ha encontrado ningún identificador de proceso.

6. Ahora se comprobara el estado de internet primero debemos estar conectado a Cassandra y luego en otro terminal.

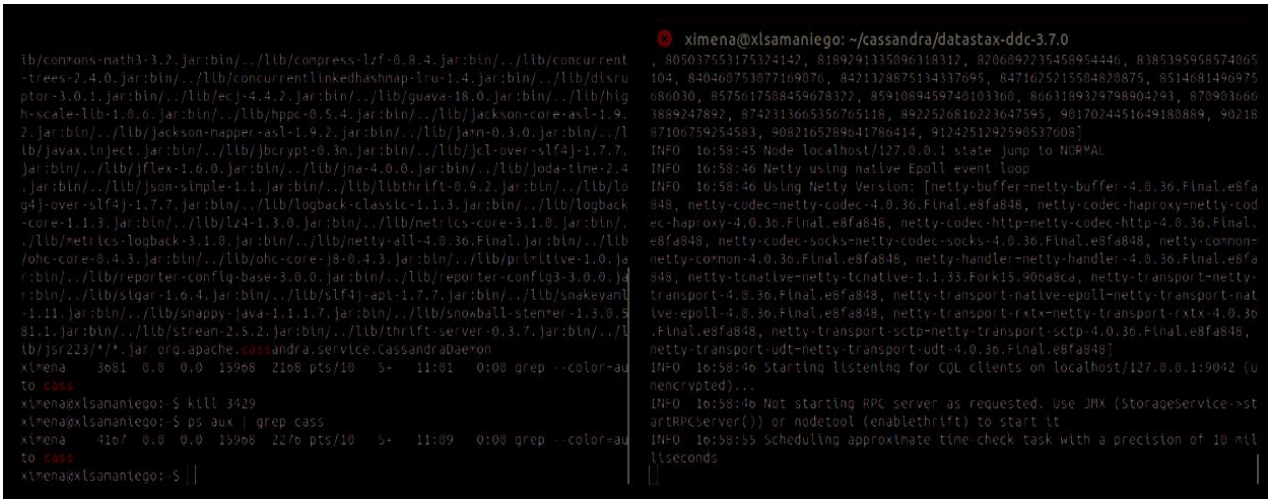


Figura 6. Ejecución de Cassandra y Búsqueda del Id proceso.
Elaboración: propia.

7. En donde se está ejecutando Cassandra colocamos lo siguiente, (cd ~/cassandra/datastax-ddc-3.7.0/),

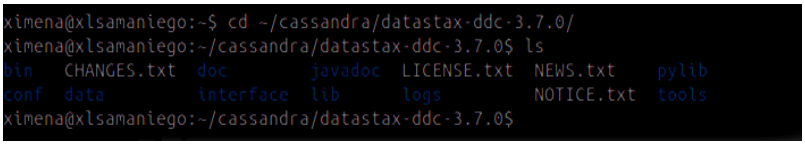


Figura 7.Creacion de Cassandra en directorio especificado.
Elaboración: propia.

8. Vamos a poder ver el estado de internet, con el siguiente comando (bin /nodetool status).

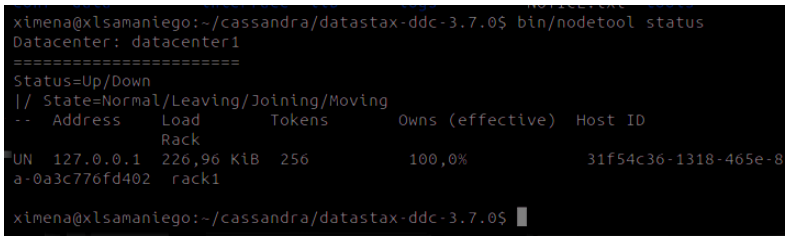


Figura 8.Estado de Internet.
Elaboración: propia.

Aquí nos muestra un centro de datos, donde se inició correctamente el nodo.

9. Por ultimo vamos a colocar este comando (bin/nodetool ring). Nos muestra el

estado de nodo y la información sobre el anillo según lo determinado por el nodo que se está consultando. Esta información le puede dar una idea del equilibrio de carga y si los nodos están abajo. Si el clúster no está configurado correctamente, diferentes nodos pueden mostrar un anillo diferente.

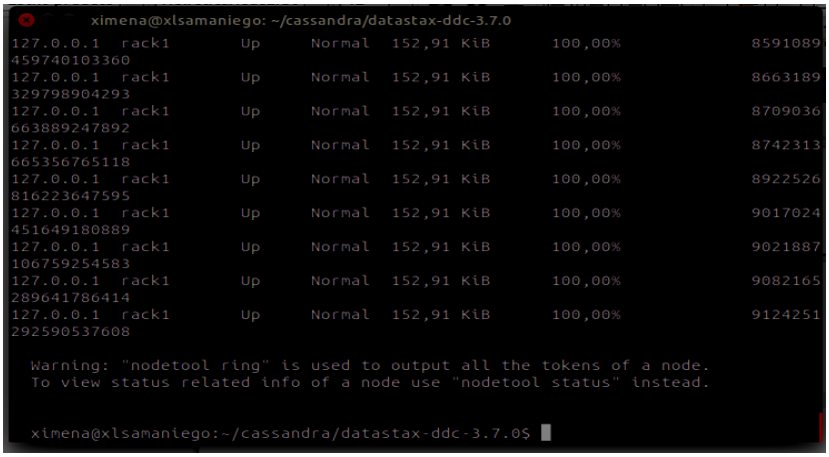


Figura 9.Estado del nodo.
Elaboración: propia.

10. Creación de Nodos en Cassandra

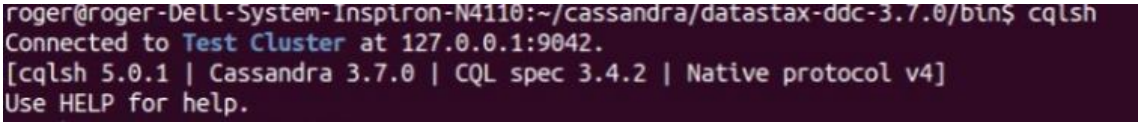


Figura 10. Conexión con cqlsh.
Elaboración: La Autora, Ximena Samaniego A.

En la siguiente figura observamos la conexión al clúster 127.0.0.1:9042 con cqlsh y las versiones en las que está funcionando.

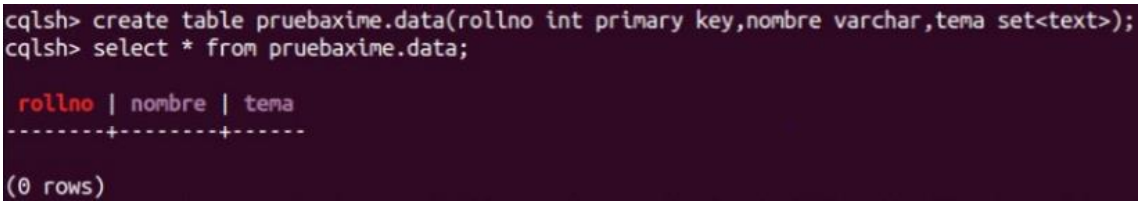


Figura 11. Create table.
Elaboración: La Autora, Ximena Samaniego A.

Creamos una tabla donde se van añadir los nodos correspondientes en cada columna de acuerdo a la keyspace creado anteriormente.

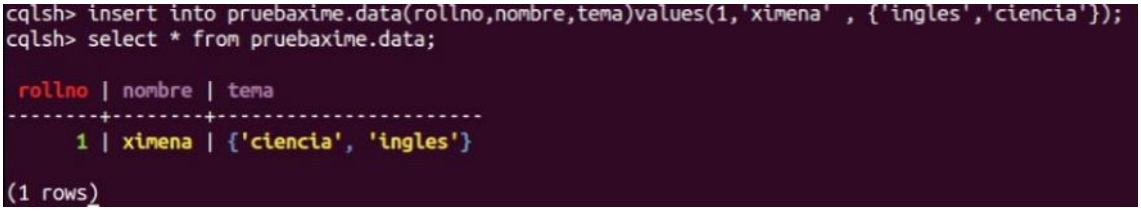


Figura 12. Inserción de datos.
Elaboración: La Autora, Ximena Samaniego A.

A continuación en la figura vamos a insertar los diferentes datos a la tabla creada anteriormente con keyspace pruebaxime.data.

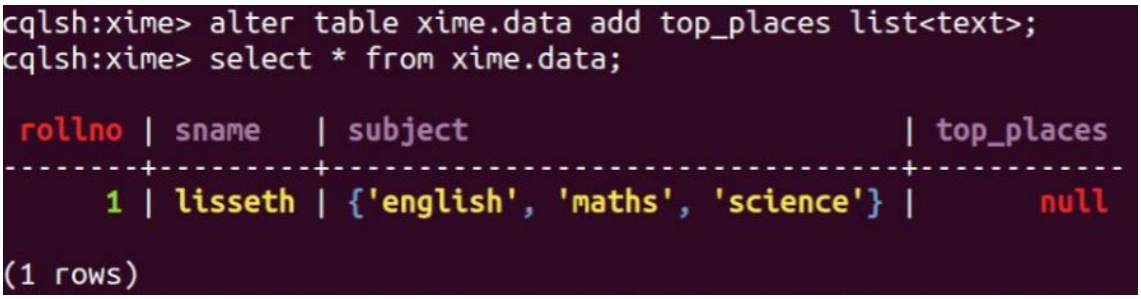


Figura 13. Edición de tabla.
Elaboración: La Autora, Ximena Samaniego A.

Hace referencia al keyspace ingresado y a continuación se añade una columna llamada top_places.

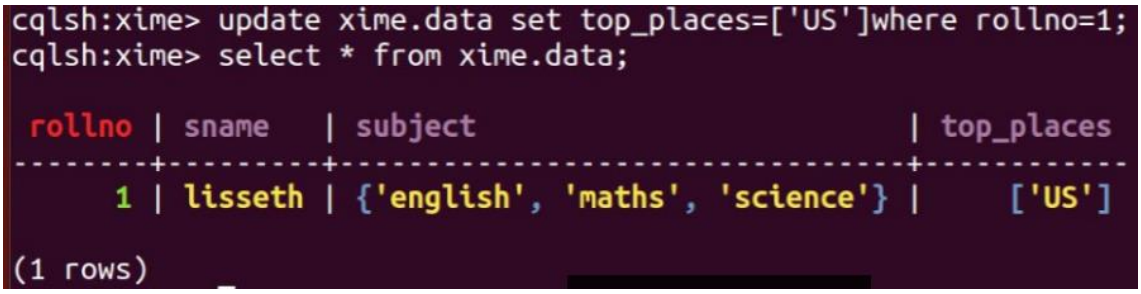


Figura 14. Actualización de tabla.
Elaboración: La Autora, Ximena Samaniego A.

Podemos actualizar la tabla agregando valores a la columna de top_places.

Anexo H: Errores y Soluciones dentro de funcionamiento de Cassandra, Zookeeper y DataDog.

Errores presentados.

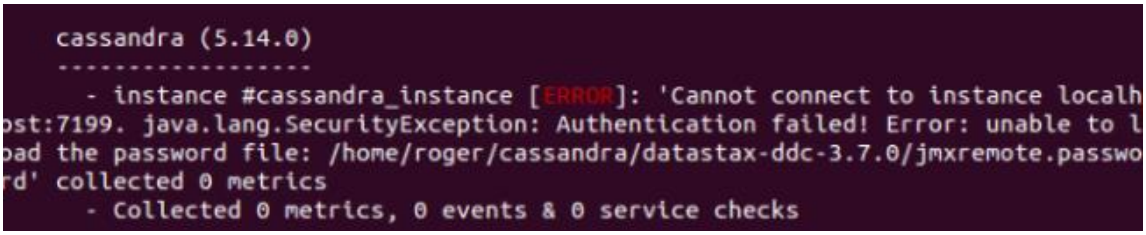


Figura 1: Error de carga de métricas de cassandra
Elaboración: La Autora, Ximena Samaniego A.

Como se puede visualizar en la figura, donde se ejecuta el servicio de DataDog info nos presenta un error de conexión por lo que no se llega a cargar las métricas correspondientes. A continuación mostramos la solución.

Figura 2: Solución de carga de métricas de Cassandra
Elaboración: La Autora, Ximena Samaniego A.

Su solución está en la configuración del archivo jmxremote.password, donde vamos añadir el user y password de cassandra y luego verificar que estén los mismo datos dentro de DataDog en la carpeta conf.d, el archivo cassandra.yaml. y luego volver a ejecutar DataDog.

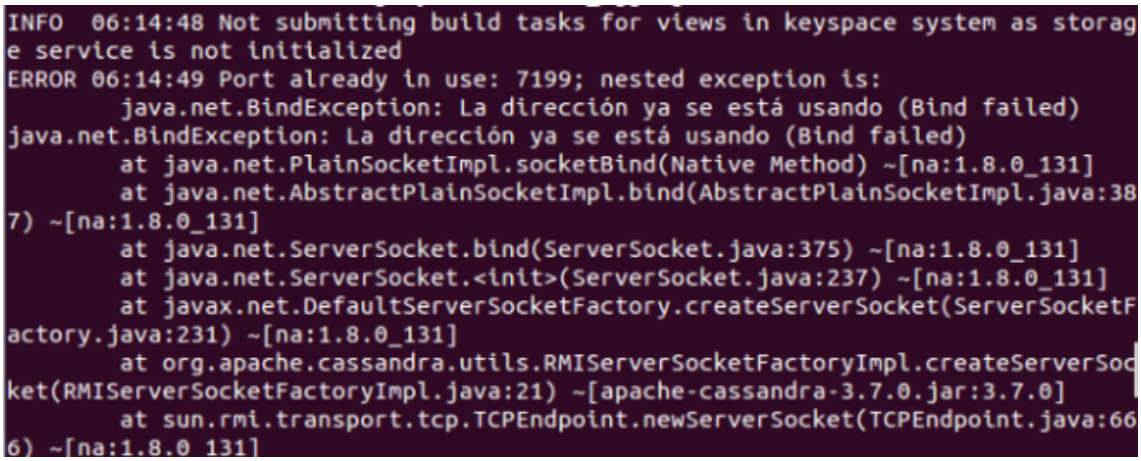


Figura 3: Error al ejecutar cassandra
Elaboración: La Autora, Ximena Samaniego A.

Cuando se presenta este tipo de error es porque java necesita más tamaño de lo establecido al momento de instalación, por lo que cuando se tiene una computadora con bajas capacidades es un poco difícil ya que se llena la memoria y nos presenta este tipo de error, a continuación se presenta la solución.

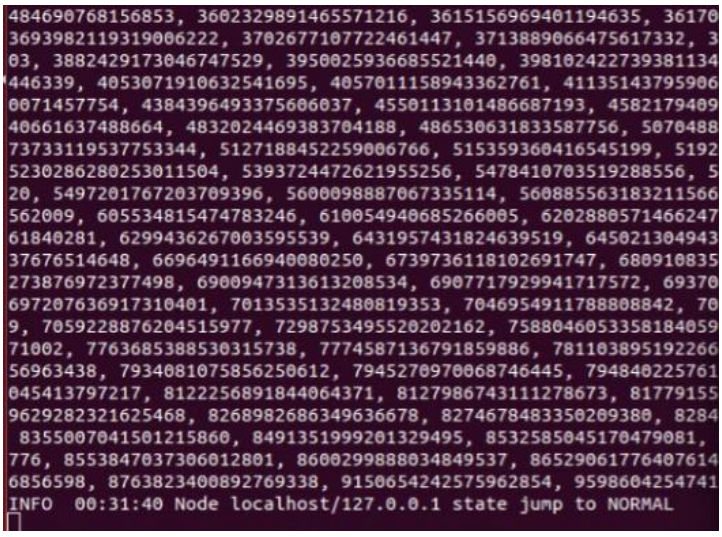


Figura 4: Solución de inicio de Cassandra
Elaboración: La Autora, Ximena Samaniego A.

La solución radica en limpiar la cache de los procesos que se están ejecutando y solo tener Cassandra iniciando, luego volver a iniciar Cassandra.

Anexo I: Integración de métricas de ZooKeeper con DataDog

Zookeeper maneja algunas métricas de DataDog con el fin de:

- Visualiza el rendimiento y utilización de ZooKeeper.
- Correlaciona el rendimiento de ZooKeeper con el resto de sus aplicaciones.

5.2.1. Configuración. Configuración del agente para conectarse a ZooKeeper, editar conf.d / zk.yaml**.**

Init_config:g

Instances:

Host: localhost

Port: 2181

Timeout: 3

Nota: Reiniciar agente.

5.2.2. Validación. Luego ejecutamos el comando info de DataDog para que me lea las métricas de zookeeper en el cual la salida debe ser así**.**

```
Checks
======

  [...]

  zk
  --
    - instance #0 [OK]
    - Collected 8 metrics & 0 events
```

Figura 24: salida de métricas de zookeeper en DataDog.
Fuente: http://docs.datadoghq.com/integrations/zookeeper/

Anexos J: Detección de fallos de Cassandra y Zookeeper.

1. Detección de fallos en Zookeeper

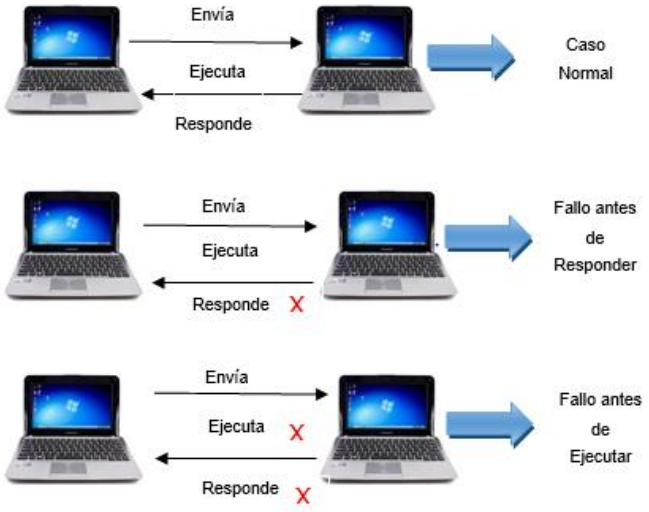


Figura 1: Detección de fallos.

Elaboración**:** La Autora, Ximena Samaniego A

En Zookeeper podemos detectar nodos fallidos con la creación de znodes efímeras, sin embargo el inconveniente principal del uso de Zookeeper es que todos los nodos deben estar conectados a la misma red,

Sin embargo, hay errores de conexión cual son normalmente los más comunes y cómo hacer para que se pueden recuperar de forma automática, se podría implementar a futuro algunos mecanismos de detección de fallos, que podrían ser usados para aumentar la estabilidad del sistema debido a la transitoria de pérdidas de conexión.

En la implementación de un sistema distribuido a futuro se debería implementar algunas técnicas tolerantes a fallos por el cual deberían prestar un diseño e mantenimiento de un sistema fallos como las cuatro frases descritas.

1. Detección de errores.
2. Diagnosticó de daños.
3. Recuperación de daños.
4. Tratamiento de fallos y servicio mantenido.

2. Tolerancia en fallos con Cassandra

Cassandra maneja una arquitectura peer to peer, en vez de una arquitectura maestro-esclavo, por tal motivo no hay puntos de tolerancia a fallos, además Cassandra puede soportar un numero grande de servidores o nodos.

En Cassandra se utiliza mucho lo que es la replicación por lo que cada dato se almacena en más de una ubicación. Esto es porque, si un nodo falla el usuario puede recuperar la información en otro lugar. Es decir se puede reemplazar nodos que presenten fallos sin tiempo de interrupción de la aplicación.

3. Predicción de fallos

Es necesario poder corregir el comportamiento y rendimiento del sistema distribuido ante posibles fallos que pueden transcurrir mediante la ejecución del mismo. No basta con asegurar su seguridad y minimizar el tiempo que el sistema no esté funcionando, sino que es necesario tener una actitud proactiva, llamada gestión de fallos.

Esta fase tiene como objetivo predecir fallos un cierto tiempo antes de que ocurran para así poder tomar medidas necesarias para evitarlos o minimizar los daños causados, por lo general este proceso lleva a cabo cuatro pasos:

- **Fallos**

Identifica posibles situaciones que el sistema pueda fallar. Normalmente este punto se lleva cabo mediante técnicas de datamining y maching learning.

- Diagnóstico: Una vez que se sepa dónde se podría producir un fallo, hay que saber dónde y por qué.

- Planificación: Definición de medidas a tomar para responder a un futuro fallo del sistema.
- Reacción: Se propone poner a ejecución el paso anterior, en este caso se basa en predecir cuándo hay un fallo o no.

Anexos K: Exploración con otras herramientas gráficas.

En el transcurso del desarrollo e investigación se ha experimentado con otras herramientas demostradas a continuación.

4. ManageEngine

Otra herramienta que nos ayuda con el monitoreo de aplicaciones tanto para Cassandra y ZooKeeper, la Aplicación Manager tiene como objetivo ayudar a los administradores a administrar su servidor Zookeeper. Recoge todas las métricas que pueden ayudar a solucionar problemas, gráficos de rendimiento de visualización y ser alertados automáticamente los problemas potenciales. Se realizó una investigación y esta solo ofrece sus servicios a aplicaciones empresariales por lo que cuentan con sistemas de seguridad que permite garantizar el uso de sus aplicaciones, en cuanto a pruebas ofrece un paquete de Login free, pero para aplicación como Cassandra es gratis por 14 días, sin embargo con ZooKeeper, recién se ha realizado cuya integración al sistema pero hasta al momento cuenta con un costo de inicio de $795, por el cual esta herramienta puede llegar ser muy útil para aquella empresa que realiza este tipo de supervisión.

5. Sematext

Sematext me permite ir más allá de simples métricas de recolección, también permite descubrir transacciones de base de datos más lentos nos muestra una infraestructura de como se está conectado en tiempo real, además permite correlaciona métricas con aplicaciones de servidor y registro de eventos, alertas, anomalías, en cuanto a la

implantación de esta herramienta de acuerdo a los recursos que vaya consumiendo y tiempo empleado tiene un costo fijo para cada acción por lo que no hubo como implementar esta aplicación. Cabe recalcar que esta herramienta puede llegar hacer muy útil ya que nos da una gráfica global de cómo van cada uno del procesos.

6. Grafana

En cuanto a Grafana, se realizó una pequeña demostración para realizar la captura de métricas ya que este es un editor de consultas de grafito avanzado el cual presentaba gráficos de alto nivel, se procedio a instalar los requerimientos de dicha herramienta siendo open source, al empezar su instalación se produjeron errores con influxdb, ya que no iniciaba dicha sección, por lo que no se podía crear la base de datos para que Grafana guardara datos acerca de sus métricas, una vez superado dichos errores, se continuo con el procedimiento, pero la página oficial de la misma no informaba con certeza las configuraciones siguientes para ZooKeeper por lo que realice una investigación donde recién se estaba implementando con ZooKeeper y aun no estaban predefinidos las configuraciones del mismo, por lo que me ayudo a tomar la decisión de que aún no podía utilizar esta herramienta, pero en un futuro podría a llegar ser muy satisfaciente utilizar dicha herramienta.

ANEXO L: API DE ZOOKEEPER EN JAVA.

API Zookeeper, posee una aplicación que puede conectarse, interactuar, manipular datos, coordinar, y finalmente desconectarse de un conjunto Zookeeper.

Se procedió a implementar una aplicación realizada en java con código ya escrito en el tutorial Zookeeper como lo descrito en la página, https://www.tutorialspoint.com/zookeeper/zookeeper_api.htm, lo cual me permitió implementar métodos de Zookeeper descritos en java para conocer acerca el funcionamiento del mismo lo único que se implemento fue la interfaz para que me mostrarse los datos dentro de la interfaz.

1. Creación de nodo.

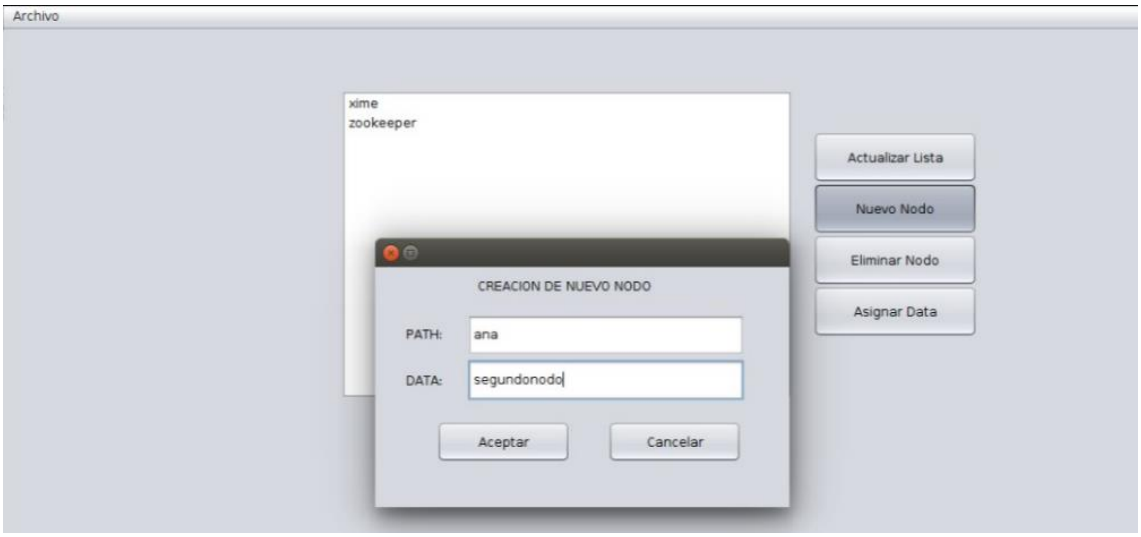


Figura 1. Creación de nodo.
Elaborado: autora Ximena Samaniego.

En la figura 1, procedemos a crear el nodo con nombre Ana, y con un mensaje de segundo luego aceptamos y el nodo se va crear.

2. Actualización de nodo.

Figura 2. Actualización de nodo.
Elaborado: autora Ximena Samaniego.

Luego en la figura 2 se continuó con la actualización del nodo, por lo que ya se puede visualizar en la interfaz.

3. Asignación de nodo.

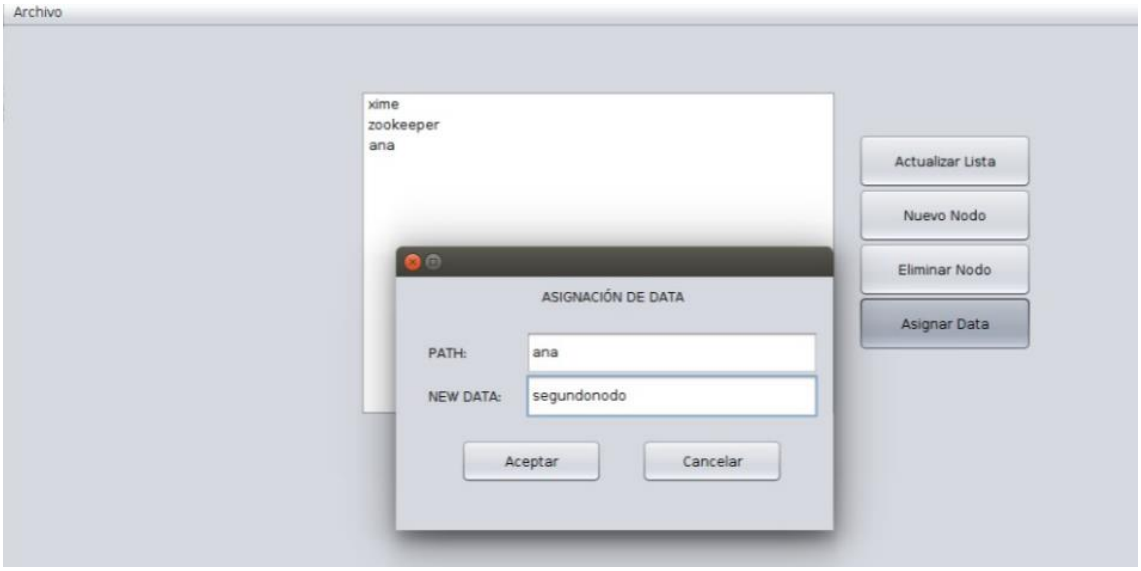


Figura 3. Asignación de nodo.
Elaborado: autora Ximena Samaniego.

Luego se procede hacer una asignación al nodo, es decir se le pondrá una descripción que es segundo nodo y se procede actualizar como en la figura 3.

4. Eliminación de nodo.

Figura 4. Eliminación de nodo.
Elaborado: autora Ximena Samaniego.

En la figura 4, se procede a eliminar el nodo creado llamado Ana, por lo que en su interfaz ya no aparecerá y todo lo que contenga el nodo.